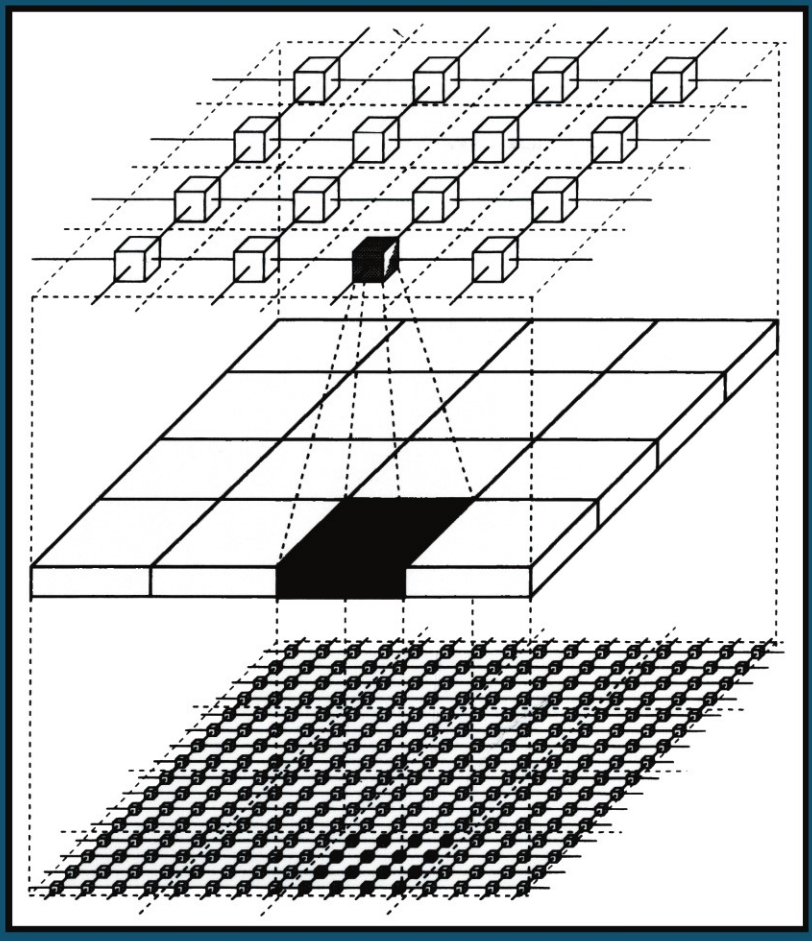
Advances in Computer Vision and Machine Intelligence

Pyramidal Architectures for Computer Vision



VIRGINIO CANTONI
AND
MARCO FERRETTI

Pyramidal Architectures for Computer Vision

ADVANCES IN COMPUTER VISION AND MACHINE INTELLIGENCE

Series Editor: Martin D. Levine
McGill University
Montreal, Quebec, Canada

COMPUTER VISION FOR ELECTRONICS MANUFACTURING
L. F. Pau

HUMAN ENGINEERING IN STEREOSCOPIC VIEWING DEVICES
Daniel B. Diner and Derek H. Fender

PYRAMIDAL ARCHITECTURES FOR COMPUTER VISION
Virginio Cantoni and Marco Ferretti

SIGMA: A Knowledge-Based Aerial Image Understanding System
Takashi Matsuyama and Vincent Shang-Shouq Hwang

A Continuation Order Plan is available for this series. A continuation order will bring delivery of each new volume immediately upon publication. Volumes are billed only upon actual shipment. For further information please contact the publisher.

Pyramidal Architectures for Computer Vision

VIRGINIO CANTONI

and

MARCO FERRETTI

University of Pavia
Pavia, Italy

SPRINGER SCIENCE+BUSINESS MEDIA, LLC

Library of Congress Cataloging-in-Publication Data

Cantoni, V.
Pyramidal architectures for computer vision / Virginio Cantoni and Marco Ferretti.
p. cm. -- (Advances in computer vision and machine intelligence)
Includes bibliographical references and index.
ISBN 978-1-4613-6023-0 ISBN 978-1-4615-2413-7 (eBook)
DOI 10.1007/978-1-4615-2413-7
1. Computer architecture. 2. Computer vision. I. Ferretti, Marco. II. Title. III. Series.
QA76.9.A73C35 1994
006.4'2--dc20 93-29212
CIP

To Laura, Vera, and Livio

—V.C.

To Elvira and Eleonora

—M.F.

Preface

Computer vision deals with the problem of manipulating information contained in large quantities of sensory data, where raw data emerge from the transducing sensors at rates between 10^6 to 10^7 pixels per second. Conventional general-purpose computers are unable to achieve the computation rates required to operate in real time or even in near real time, so massively parallel systems have been used since their conception in this important practical application area.

The development of massively parallel computers was initially characterized by efforts to reach a speedup factor equal to the number of processing elements (linear scaling assumption). This behavior pattern can nearly be achieved only when there is a perfect match between the computational structure or data structure and the system architecture. The theory of hierarchical modular systems (HMSs) has shown that even a small number of hierarchical levels can sizably increase the effectiveness of very large systems. In fact, in the last decade several hierarchical architectures that support capabilities which can overcome performances gained with the assumption of linear scaling have been proposed. Of these architectures, the most commonly considered in computer vision is the one based on a very large number of processing elements (PEs) embedded in a pyramidal structure.

Pyramidal architectures supply the same image at different resolution levels, thus ensuring the use of the most appropriate resolution for the operation, task, and image at hand. Furthermore, a hierarchy is introduced by interplane communication which allows the implementation of a general ''planning'' strat-

egy. Here the approach is to solve problems at a low spatial resolution, and therefore with a small amount of data, and then to proceed to successive refinements until the final verification of the results at the highest resolutions available. In this way, speedup factors greater than the number of PEs can be obtained in the quoted low-level vision tasks, because data reduction is an exponential function of the number of levels which have been used.

This approach has shown how a few hierarchical levels can substantially increase the processing efficiency of systems comprising many PEs; this behavior pattern has found a precise verification in various practical problems. Even in nature several large systems, composed of a great number of elements, have been shown to be self-organizing in a hierarchical structure, and the rules that these large sets configure follow the theory of HMSs.

This book is split logically into three sections, which consist of Chapters 1 and 2, Chapters 3 to 8, and Chapters 9 and 10, respectively.

The first section provides the groundwork for the architectural analysis carried out in the remainder of the book. It discusses the role of the hierarchy in setting up complex systems and then specializes in the use of the hierarchy in computer vision systems.

Chapter 1 reviews the theory of hierarchical modular systems, which model the behavior of large self-organizing natural systems, such as monetary systems, settlement distribution over a territory, and natural languages. This theory shows that the introduction of a few hierarchical levels substantially increases the effectiveness of such large systems. The modularity criterion is inherited to some extent in the structure of most hierarchical architectures.

Chapter 2 discusses in depth the benefits of hierarchical strategies in the vision domain. Essentially, the motivation for using such an approach is that of obtaining high computational performances by processing only the relevant image data at the right time. The behavior of the known parts of the human vision system is used as a guideline. Its *preattentive* and *attentive* phases perform the basic tasks of any hierarchical processing paradigm—that is, delineating the region of interest and focusing on it for a detailed scrutiny. In computer vision, such a paradigm is supported by ad hoc data structures that consist of *multiresolution* grids, which reproduce the image in different amounts of details. The term *pyramid* is used to clarify the hierarchical connections between adjacent grid layers. This chapter analyzes the alternatives for building such representations and the possible processing strategies.

The second section more closely covers the architectural issues. It first introduces a framework for comparing the possible solutions according to topology and to its functional composition. Then it describes and discusses the

actual machines and/or prototypes that can be described as pyramid architectures or that have a pyramid processing mode explicitly supported in hardware. The simulation of pyramids on other parallel systems is also covered in detail.

Chapter 3 describes all system organizations that construct a hierarchy with a homogeneous set of processing elements. In such cases, the topology of the interconnections is the main feature of the resulting system. The families of hierarchical systems considered include snowflakes, stars, trees, hypernets, and pyramids. A quantitative assessment of these hierarchies is carried out through a set of parameters that measure the capability of the network to sustain data exchanges among the processing elements.

Chapter 4 focuses on hierarchical machines that have already been built (or at least fully designed) and organizes them into a taxonomy. This taxonomy is itself a small hierarchy with two levels. The first level splits the systems into homogeneous or heterogeneous ones according to their processing module capability. The second is based on the means of coupling these modules and on interconnection networks (tight–loose, compact–distributed, fixed–reconfigurable). The processing paradigm varies within the taxonomy: these include pipeline, SIMD, multi-SIMD, and MIMD systems.

Chapter 5 concentrates on the most popular hierarchical topology, the basic pyramid, and on the homogeneous, massively parallel systems that have been proposed or at least built in prototype. When designing a pyramid structure, one may follow two approaches: the first with fine granularity, where one processor per image pixel is conceived, and the second with coarse granularity, where one microprocessor is associated with an image block. The former approach is the one followed in most cases, perhaps because of the expected benefits in designing a parallel system with VLSI. This chapter offers a comprehensive and in-depth analysis of the actual *pyramid computers* that attracted so much interest in the mid- and late-1980s.

Chapter 6 analyzes alternatives to the true pyramid computer. Specialized hardware solutions have been proposed to achieve multiresolution processing without resorting to these expensive parallel architectures. Pipelined systems specializing in decimation (pyramid building) and expansion are the most effective alternative. The processing facilities of such systems offer a powerful multiresolution environment that can be easily integrated into low-cost, application-specific devices.

Chapter 7 addresses a more pragmatic issue. Massively parallel systems, once a small niche in the computer processing community, have now come of age, and commercial systems have become more and more widespread. Since none of them adopts a pyramid topology, it is worth studying the cost of em-

bedding pyramids into their native structure. The mesh and the hypercube are the two most common topologies of such commercial systems. This chapter reviews embeddings proposed for the basic pyramids in these topologies.

Chapter 8 closes the central section of the book by analyzing heterogeneous hierarchical systems. The rationale for designing a hierarchical system according to this paradigm is that it is difficult to match the computational requirements of the various processing steps (low to high) in a vision problem with just a single homogeneous architecture. Low-level tasks demand specialized hardware capable of matching the high speed of incoming data. Subsequent processes are best matched onto coarse-grained standard microprocessors. Some systems have been designed to merge the pyramid concept with this heterogeneous structure.

The third section covers the user's point of view. Programming tools, including languages and development tools, are notoriously the most difficult part of a system project. With the recent advances in VLSI design, the time required to specify, design, build, and assemble a functioning prototype is shorter by order of magnitudes than the corresponding time to obtain a standard compiler. However, effective use of the hierarchical system must be made possible to the ''naive'' end user, whose energy should only be concentrated on the tasks, with almost no regard to the intricacies of the complex system he or she is programming.

Chapter 9 covers the language side of this problem. Specific attempts have been made to conjugate standard languages with the pyramid architecture. Both the data structures and the semantics of the control operators need a revised interpretation.

Chapter 10 faces the ultimate question of multiresolution processing. The expected advantages of this processing strategy must be measured (both from a theoretical computational point of view and in practical situations) and proved to be relevant. The chapter contains a selection from the huge number of algorithms that exploit pyramid processing in very diverse computer vision contexts. This overview, which does not pretend to be an exhaustive and up-to-date tutorial on the subject, focuses on those algorithms which apply the multiprocessing strategy at its best.

Virginio Cantoni
Marco Ferretti

Pavia, Italy

Acknowledgments

Many are the people who contributed to the effort of setting up this book. Some kindly provided up-to-date material on their work; others simply helped informally with suggestions. Among them, we would like to thank particularly Dr. Angelo Buizza for his guidance through the biological aspects of human vision, Dr. Mauro Mosconi, and our co-workers in the Computer Vision and CAD Laboratories of Pavia University.

Contents

Pyramidal Architectures for Computer Vision

Chapter 1

Hierarchical Architectures

The theory of hierarchical modular systems (HMSs) has shown how a paltry number of hierarchical levels can massively increase the efficiency of very large systems. Several large natural systems have been identified as self-organizing HMSs, including monetary systems, distribution of settlements on a territory, natural languages (hierarchy given by letters, syllables, words, predicates, clauses, sentences, and paragraphs), military hierarchies, etc. In this chapter, the theory of HMS is briefly introduced, a few examples are described, and hierarchical computer architectures are introduced within the framework of the HMSs.

1.1. INTRODUCTION

The notion of a system's architecture is somewhat more than the elements which are used and the rules for their composition. An example is the design of arches: the set of bricks placed one next to another in a curve to form an arch gives rise to a new capacity of supporting weights that is not given simply by addition to the elementary property of each component: a well-structured *system* comes forth, able to overstep the crude joining of the parts. Arch designing points out directly this astonishing capability to bring forth new qualities not directly derived from the scaling of the ingredients.

This precise aspect earned for architecture, until the Renaissance, the rep-

utation of *ars regia*. Its typical tools such as a pair of compasses, plumb line, and level, became the emblems of universal order, the keys of cosmic models. Even in Islamic culture, as can be discovered in the main buildings (and particularly in places of worship), this static paradox has been used to extremes: the sustaining parts are minimized and multiplied, intermingling the rarefied and delicate arches in a kaleidoscopic mode and covering larger and larger tracts. Examples of these buildings are spread over all lands touched by Moslems, from Spain to Sicily to Pakistan. Let us cite as a masterpiece of this kind of architecture the Mosque of Cordoba. This very example constitutes a paradigm which can be applied to computer vision. The technique employed in the Mosque of Cordoba, expressed in computer terms, involves nothing other than minimizing and multiplying elementary modules to support increasing computing demand (e.g., larger portions of the visual field). This system was indeed applied in one of the most popular hierarchical architectures for computer vision: the compact, fine-grained pyramid.

Computer vision deals with the problem of extracting useful information from large quantities of sensory data: raw data emerges from the transducing sensors at rates ranging from 10^6 to 10^7 pixels per second. Conventional general-purpose computers are unable to achieve the computation rates required to operate in real time, or even in near real time, in processing images.

The development of special-purpose architectures for computer vision has been previously characterized by efforts to reach, for typical algorithms of the preprocessing stages of the image arrays, a speedup factor equal to the number of processing elements (linear scaling assumption according to Hewitt and Lieberman[1].) In this approach, maximum efficiency (Siegel *et al.*[2]) is unity when the ideal speedup (equal to the number of processing elements, PEs) is reached. This behavior pattern can nearly be achieved only when there is a perfect match between computational structure or data structure and system architecture.[3]

In recent decades, some architectures that support capabilities which overcome the performance gained with the assumption of linear scaling were proposed; among them, the most commonly considered in computer vision is based on a very large number of PEs embedded in a pyramidal structure.

Pyramidal architectures supply the same image at different resolution levels, thus ensuring the use of the most appropriate resolution for the operation, the task, and the image at hand. Furthermore, a hierarchy is introduced by the interplane communication, which allows the implementation of a general ''planning'' strategy (Kelly[4] and Tanimoto[5]), in which the approach is to undertake the solution of problems at low spatial resolution and to proceed to successive refinements till the final verification of the results is reached at the

highest resolutions available. In this way speedup factors greater than the number of PEs can be obtained even in the low-level vision tasks mentioned, because data reduction is an exponential function of the number of levels that have risen.

This approach has shown how a narrow number of hierarchical levels can substantially increase the processing efficiency of systems composed of numerous PEs; and even though this strategy has not yet been adequately formalized, it has found precise verification in various practical problems, including object recognition, implementation of multigrid techniques in numerical analysis, relaxation algorithms, matrix operations, transient simulation in electrical circuits, and topographical problems.

Even in nature several large systems, composed of a very great number of elements, have been identified as self-organizing in a hierarchical structure, and a theory has been developed to explain how these large sets configure themselves.

1.2. THEORY OF HIERARCHICAL MODULAR SYSTEMS

A hierarchical system of n *levels* ($0 \leq n \leq \infty$) is composed of n disjoint sets of elements. Let us call c_h the number of elements belonging to the level h ($0 \leq h \leq n$); the total number N of elements is then

$$N = \sum_{h=0}^{n} c_h \tag{1.1}$$

The arrangement defined by $\{c_0, c_1, c_2, c_3, \ldots, c_n\}$ is called a *partition* Π of the elements of the system.

Let us suppose that a real number v_h is assigned to each element of a level h, called the *value* associated with that level. Hence, the total value V of the system is

$$V = \sum_{h=0}^{n} c_h v_h \tag{1.2}$$

and the average value $\langle v \rangle$ is

$$\langle v \rangle = \sum_{h=0}^{n} c_h v_h \Big/ N \tag{1.3}$$

The hierarchical system is called modular, and M is the *module* of the system, when the following relationship holds between any pair of consecutive levels:

$$\frac{v_h}{v_{h-1}} = M > 1 \qquad \forall h, \qquad 1 \leq h \leq n \tag{1.4}$$

in this case, $v_h = v_0 M^h$.

A partition Π_b of a hierarchical modular system is called a *refinement* of a partition Π_a if and only if Π_b has more levels than Π_a and if Π_b retains all the levels of Π_a. In this case, it is easy to verify that the following relationship holds between the modules of the two partitions:

$$M_b = [M_a]^{1/p} \tag{1.5}$$

where p, an integer greater than 1, corresponds to the ratio between the number of levels:

$$n_b = p n_a \tag{1.6}$$

Moreover, provided that the number of elements employed at each level is always $c_h < M$, it is easily shown that every value $\bar{V}$ that is a multiple of v_0 and not greater than the maximum value of the system ($v_0[M^{n+1} - 1]$) can be obtained by only one partition $\bar{\Pi}$:

$$\bar{V} = v_0[\bar{c}_0 + \bar{c}_1 M^1 + \bar{c}_2 M^2 + \cdots + \bar{c}_n M^n] \tag{1.7}$$

This result is well known and easily recognized in the numerical system theory (e.g., decimal system); in fact, in this case the above-mentioned conditions are satisfied.

These HMSs defined according to the aforementioned conditions retain an invariant factor γ under refinement transformations: for given M, n, and p the ratio between the number of possible partitions and the module is constant:

$$\Gamma = \frac{M^{n+1}}{M} = \frac{(M^{1/p})^{pn+1}}{M^{1/p}} = M^n \tag{1.8}$$

Subsequently the aforementioned properties are highlighted in a very common case which strictly follows the HMS theory. In fact, Caianiello[6] identifies sev-

eral large natural systems as HMSs, among which are monetary systems. The first problem we address is: why introduce a number of currencies, and what are the best values to assign them? Here, the main intention is to be able to buy goods, in a given range $(1–M_v)$ of values with a uniform distribution and with few tokens. The minimum number of tokens is obviously obtained when all the values are minted ($\bar{n}=1$); it is maximum ($\bar{n}=[M_v+1]/2$) adopting just one currency. It is quite evident that by introducing tokens of different values, we obtain an $\bar{n}$ between the quoted extremes: in particular, if the requested values are uniformly distributed in the range $1–M_v$, then the higher the number of different tokens, the lower $\bar{n}$ can be.

It is easy to show (details of the proof can be found in Caianiello *et al.*[7] that the average number of tokens necessary to compose all the values from 1 to M_v is a minimum if the monetary system is modular, that is, if the values of the tokens follow the law given in Eq. (1.4). In this case, the maximum number of tokens of each type that can occur is $M-1$. Consequently, if $(M-1)/2$ is the average number of tokens for each value, the average number $\bar{n}$ for all tokens will be

$$\bar{n}=n\frac{M-1}{2} \tag{1.9}$$

When working with the decimal system, a practice universally adopted, a good module can be 10 (in effect, bank notes of value . . . 1, 10, 100, 1000, . . . are used in all monetary systems). Nevertheless, in practice a higher number of levels is always introduced in order to further decrease $\bar{n}$. This refinement is usually given for $p=3$ (three mints for a decade ratio); that is, the most common module is $M=\sqrt[3]{10}$, to which corresponds the sequence of currencies

$$1, 2.15, 4.64, 10, 21.5, 46.4, 100, 215, 464, \ldots$$

which is usually rounded to the nearest integer at the most significant digit. Even if a few of these values are missing in some currencies, everyone would recognize the corresponding monetary sequence:

$$1, 2, 5, 10, 20, 50, 100, 200, 500, \ldots$$

Since it is well known, this set of currencies is suitable for satisfying the property expressed in (1.7). Moreover, the average number of tokens needed

to compose the generic value in a quoted range, given in Eq. (1.9), holds as well. For example, for the range $1-10^6-1$, it is 10.35 (we do not consider the case, very common in practice, in which more than M identical tokens are used: e.g., a coin of value 100 is often replaced by two coins of value 50).

1.3. SELF-ORGANIZING HIERARCHICAL MODULAR SYSTEMS

The systems considered belonging to the HMS family can adjust themselves in order to match the mandatory conditions of the external universe with which they interact: for this reason they are called *self-organizing*.[8]

The analysis which follows considers fixed the values v_h, associated with the possible levels, and calculates element distribution, which only reflects the way the system self-adjusts to external requirements.

A second assumption refers to the way in which the interaction with the external environment is realized: let us suppose that this interaction is of a "global" nature; i.e., the requirement is not made directly on internal distribution but can be expressed only by statistical parameters of the whole HMS, such as management of the average value.

Under these hypotheses, two behavior patterns can be considered: a system *evolution* and a system *revolution*. In the former case the element distribution changes, varying the partition Π of the elements of the system but not modifying the module and the number of levels. In the latter case a structural change occurs and, under the hypothesis of modularity, a partition refinement takes place.

In what follows, the analysis is limited to the latter case; a few comments about the equilibrium partition, and the relationship between levels (and associated values v_h) and populations (c_h) of a self-organizing HMS will be derived at the end.

1.3.1. The Invariance Principle and the Law of Distribution

Let us consider the case, verified by several common HMSs, in which the global average value of the elements of a system is invariant under refinement transformations (invariance principle):

$$\langle v \rangle = \frac{v_0 \sum_{h=0}^{n} c'_h M^h}{\sum_{h=0}^{n} c'_h} = \frac{v_0 \sum_{h=0}^{pn} c''_h M^{h/p}}{\sum_{h=0}^{pn} c''_h} \tag{1.10}$$

where c' and c'' are, respectively, the number of elements before and after the p-refinement [see Eq. (1.5) and (1.6)].

We can easily check that the following relationship satisfies Eq. (1.10) (in Ref. 7 this solution is shown to be unique):

$$c_h'' = c_0'' M^{-h/2p} \qquad \forall h, \qquad 1 \le h \le n \tag{1.11}$$

and, in particular, between two consecutive levels, the ratio between the number of elements is given by

$$c_h'' = c_{h-1}'' \, M^{-1/2p} \qquad \forall h, \qquad 1 \le h \le n \tag{1.12}$$

Substituting these results in Eq. (1.10) gives the invariant average value $\langle v \rangle$ of the system elements:

$$\langle v \rangle = \sqrt{\gamma} = M^{n/2} \tag{1.13}$$

Note that, as expected, $\langle v \rangle$ is independent of the refinement parameter p.

In particular, when N is constant (in addition to M and $\langle v \rangle$, mentioned before), for a given initial number of levels n and a p-refinement, the *distribution law* of the elements between the various levels which secures the invariance principle is

$$c_h'' = \frac{N}{M^{h/2p}} \frac{1 - M^{-1/2p}}{1 - M^{-1/2p - n/2}} \qquad \forall h, \quad 0 \le h \le n \tag{1.14}$$

The case $p = 1$ shows the distribution of a system at the "equilibrium" of the possible refinement "evolutions" (in Caianiello[6, 9] an analogy with physics is introduced and a "thermodynamic" explanation of the distribution law is derived). In this stable condition (a "revolution" could of course lead to another stable status), no matter what the history of the system levels was, the number of elements belonging to a level h equals the number of elements of the level immediately below reduced by a constant factor given by the square root of the module; from Eq. (1.12), c_h is then

$$c_h = \frac{c_{h-1}}{\sqrt{M}} \qquad \forall, \qquad 1 \le h \le n \tag{1.15}$$

in this case, $c_h = c_0 M^{-h/2}$.

Likewise, from Eq. (1.14), we can derive how N elements, on the basis of the invariance principle, would distribute in n modular hierarchical levels:

$$c_h = N \frac{\sqrt{M^{n+1-h}} - \sqrt{M^{n-h}}}{\sqrt{M^{n+1}} - 1} \quad \forall h, \quad 0 \le h \le n \tag{1.16}$$

1.3.2. Some Examples of HMSs in Society

Several examples of systems composed of millions of elements that self-organize, following the theory of the HMSs, can be recognized in nature; in the literature those involving humans in particular are analyzed (according to Becker[10]) the social substructure itself follows the theory of the HMSs). A short introduction to the most popular HMS family is given, namely monetary systems, distribution of settlements in a territory, and natural languages.

1.3.2.1. Monetary Systems

As mentioned, monetary systems follow the theory of HMSs quite closely. From Eqs. (1.4) and (1.15), the total value of currency A_h at a given level h is

$$A_h = c_h v_h = c_0 M^{-h/2} v_0 M^h = c_0 \sqrt{v_h} \tag{1.17}$$

This result, experimentally discovered by Hentsch[11] by analyzing the distribution of currencies in many countries (Switzerland, France, Holland, Germany, and the United States), holds for all countries. Caianiello[8] has derived Eq. (1.17) from the theory of HMSs and has confirmed the result for other countries. Furthermore, when this law, occasionally, does not hold, there is clear evidence that justifies the deviation (e.g., lack of values in the sequence, typically tokens with 2 10^x; silver coins that are being withdrawn from circulation and are so requested by coin collectors, etc.).

For example, in Figure 1.1 we show the money circulating in Italy on December 31, 1990, precisely the total amount for each currency A_h versus the value v_h. Note that the dispersion around the regression line (straight line), which is close to the Hentsch law (in log-log scale: $\ln A_h = k + 0.5 \ln v_h$), can be easily explained. The coins in the first decade are practically obsolete, and the last denominations are overminted in most monetary systems to support the missing higher denominations. The deviations given by 20, 2000, 20000 lire, not regularly minted by the "Zecca" (state mint), caused the overdistribution of the adjacent minor denominations.

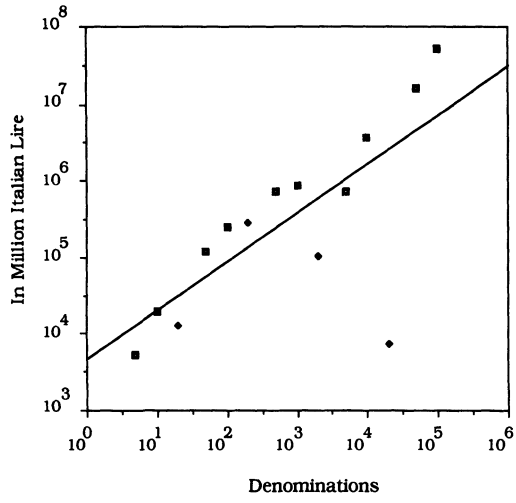


Figure 1.1. Circulating money in Italy on December 31, 1990. Denominations 5 10^x, and 10^x are represented by squares, meanwhile denominations 2 10^x, not regularly minted, are represented by diamonds. The regression line (straight line) is quite close to the Hentsch law.

1.3.2.2. Distribution of Settlements in a Territory

In 1936, Singer[12] discovered a statistical regularity in urban settlement: the higher the number of inhabitants, the lower the number of municipalities, following this equation:

$$\ln y = \mu_0 + \mu_1 \ln x \tag{1.18}$$

where x is the number of inhabitants, y is the number of towns with a number of inhabitants higher than x, and μ_0, μ_1 are coefficients that were shown by Allen[13] to be stable over several decades.

In monetary systems the number of levels n, and the values v_h associated with each level, are well known *a priori*. This is not so for a population distribution over a territory, even if it is quite evident that in cities with a higher number of inhabitants on average there is a better quality and a higher standard of living (in terms of the quantity and quality of services in the cultural, medical, administrative, marketing, etc., area present in the town).

Caianiello *et al.*[8] and Scarpetta and Simoncelli[14] showed how the Singer law can be derived by applying the refinement invariance principle. This analysis is based on the assumption (experimentally derived by analyzing the data of an Italian region) that the average number of "local units" of services and the average number of workers are proportional to the number of inhabitants.

Therefore, their product (''which gives us the value of the town'') is proportional to the square of the average population ($v_h = k\langle c_h \rangle^2$).

The comparison (introducing $n = 12$ levels of towns) with the real distribution of the population has been shown to be true for several developed countries. Even in this case the deviations can be easily interpreted politically (e.g., France, where centralism is evidenced by the absence of level $n-1$ and a smaller value for c_{n-2}). In general, the paradigm does not hold for developing countries, and in this case the deviations can be justified.[14]

1.3.2.3. Natural Languages

A third aspect of human societies, in which the larger the group the more complex it becomes, is communication among members. Much research has been done on the hierarchical organization of natural languages. Scarpetta and Simoncelli[14] showed how natural languages belong to the family of HMSs. The hierarchy was classified as letters, syllables, words, predicates, clauses, sentences, paragraphs. Some difficulties arise in defining the values associated with each level. One suggestion is that a coarse assignment can be based on the average number of letters p required to make up the element of each level. On a first training set consisting of newspaper articles, they discovered that a fair agreement between data and the theory of HMSs is obtained with $v_h = \langle p^2 \rangle^h$, where p^2 acts as the module of the HMS.[14, 15] This assumption has been shown to hold even outside the training set.

In fact, the kind of hierarchy introduced by natural languages has a quite different nature from the one belonging to monetary systems and the distribution of settlements: the elements of a level recursively constitute the components of each successive level. Becker[15] called the first a *horizontal hierarchy,* and the other case, in which level components are disjointed, has been called *vertical hierarchy*. We generally consider this last category.

1.4. HIERARCHICAL ARCHITECTURES FOR PARALLEL PROCESSING MACHINES

Becker[15] asks whether the structures that have been selected in the course of evolution, because of their efficiency, are of any value for parallel computer architectures. It has been pointed out that one of the structures can be found in several diverse systems: the modular hierarchy. In fact, the hierarchical levels supply efficiency in large systems, and modularity allows us to handle values which range by orders of magnitude with a restrained number of levels.

Two types of hierarchy can be considered: vertical (one over groups) and horizontal (grouping of groups). Solutions belonging to each of these families have been proposed and, in many cases, applied to computer vision. Hierarchical topologies are discussed in depth in Chapter 3; stars (Section 3.4.2), trees (Sections 3.5.1–3.5.3), and pyramids (Section 3.5.4) are examples of the first type of hierarchy, and snowflakes (Section 3.4.1) and hypernets (Section 3.5.5) are examples of the second.

In the first case several modules have been used; generally, the most popular cases are the bin and quad trees (pyramids) in which n_k/n_{k+1} is equal to 2 and 4 respectively. Obviously, the corresponding modules are 4 and 16. As a consequence, the higher the level the more powerful the processing element has to be in terms of CPU capability, local memory, etc. (The quantitative ratio between adjacent levels must be 4 and 16). As we will see, the different levels in the hierarchy correspond to different levels of abstraction on data, different computational paradigms (from numeric to symbolic processing), and different flexibility and concurrence capabilities. This trend is quite evident in all the hierarchical systems described in Chapter 8. In other cases technological constraints, fault tolerance limits (against processing element specialization), and complexity in handling and balancing heterogeneous components partially hide the HMS structure.

In horizontal hierarchies $\sqrt{M}$ elements of one level form one element of the successive level, and in a self-similar way $\sqrt{M}$ groups of this level form one of the next, recursively. By construction all the elements are equal. This results in a general hierarchy which does not follow the HMS paradigm. This approach, as we will see, has been followed for coarse-grained systems and is effective for high-level computations. It not a suitable approach for very large systems of simple elements which cannot be loaded with interlevel activities.

REFERENCES

1 C Hewitt and H Lieberman, Design issues in parallel architectures for artificial intelligence, MIT A I Memo No 750, pp 1–14 (1983)

2 L J Siegel, H J Siegel, and P H Swain, Parallel algorithm performance measures, in *Multicomputers and Image Processing* (L Uhr, ed), pp 241–252, Academic Press, New York (1982)

3 V Cantoni and S Levialdi, Matching the task to a computer architecture, *Comput Vision, Graphics Image Process* **22**, 301–309 (1983)

4 M D Kelly, Edge detection in pictures by computers using planning, in *Machine Intelligence*, Vol 6, pp 397–409, Edinburgh University Press (1971)

5 S L Tanimoto and A Klinger, *Structured Computer Vision Machine Perception through Hierarchical Computation Structures*, Academic Press, New York (1980)

6. E. R. Caianiello, Some remarks on organization and structures, *Biol. Cybernet.* **26**, 151–168 (1977).
7. E. R. Caianiello, G. Scarpetta, and G. Simoncelli, A systematic study of monetary systems, *Int. J. Gen. Syst.* **8**, 81–92 (1982).
8. E. R. Caianiello, M. Marinaro, G. Scarpetta, and G. Simoncelli, Structure and modularity in self-organizing complex systems, in *Topics in the General theory of Structures* (E. R. Caianiello and M. A. Aizerman, eds.), pp. 5–57, D. Reidel, Dordrecht (1987).
9. E. R. Caianiello, A thermodynamical approach to hierarchical self-organizing systems, private communication, seminar delivered at IIASA (1979).
10. J. D. Becker, Structure, justice, and efficiency, Proc. Workshop on Modelling Processing of Structural Change in Social Systems, Beiträge zur Sicherheitspolitik Nr. 3. Max-Planck-Ges., Max-Planck-Institut (1988).
11. J. C. Hentsch, La circulation des coupures qui constituent une monnaie, *J. Soc. Statist. Paris* **4**, 279–286 (1973).
12. H. W. Singer, The "courbe des populations". A parallel to Pareto's law, *Econ. J.* **46**, 254 (1936).
13. G. R. Allen, The "courbe des populations". A further analysis, *Bull. Oxford Inst. Statist.* **16**, 179 (1954).
14. G. Scarpetta and G. Simoncelli, Self-organizing hierarchical modular systems, in *WOPPLOT 86—Parallel Processing: Logic, Organization and Technology* (J. Becker and I. Eisele eds.), pp. 87–119, Springer-Verlag, Berlin (1987).
15. J. Becker, Structural aspects of organizing parallel processing machines, K. Ecker (Hrsg.), Berichte des Instituts für Informatik der Universität Clasthal (1988).

Chapter 2

Hierarchical Strategies in Computer Vision Systems

In order to achieve the high performance that real applications require, correct computation on the relevant image data at the right time is essential. Following the studies on vision and perception in humans, two phases can be distinguished: (1) a *preattentive* phase, in which the visual system is only dedicated to the detection of events and regions of interest within its wide field of view, and (2) an *attentive* phase, in which an extensive analysis of a restricted amount of data is performed. Correspondingly, an equivalent computational paradigm will be introduced in order to reduce the huge amount of raw data transduced by a standard artificial vision sensor. Such a paradigm provides for the use of variable-resolution grids, according to the image detail required for the task, thus obtaining *multiresolution* systems with different-sized layers.

2.1. INTRODUCTION

It has been explicitly stated by many authors[1–3] that vision in humans, and even in higher animals, is not a pure mechanical act but a creative one. The discovery of what is present in the scene, and where it is, is usually achieved by means of a ''smart'' selection of data located almost anywhere in the wide field of view, even at an early stage of analysis. Human vision has a sophisticated control mechanism that can move and ''focus the camera eye in these regions.''

A practical example of the effectiveness of this process is object inspection during manufacturing. In this case the object may be large and intricate, but a human inspector quickly locates just those points that need close examination. When automated, visual inspection involves two distinct and sequential tasks as well:[4] (i) locating the object to be inspected; (ii) the detailed inspection of the component to check its compliance to specifications. Freeman[4] clearly describes this process: ''When carried out by a human inspector, the first of these (steps) is usually done subconsciously; in a machine vision system it must be addressed as an explicit task that is often the most difficult part of inspection.''

In order to achieve performance comparable to that of a human inspector, automatic systems must be able to implement some kind of coarse-to-fine homing procedures and closely examine only a restricted subset of the input data. This is the only way to reduce the overall computational cost by any order of magnitude.

The basic elements needed to implement this process are discussed in this chapter. The selection in space and time of events and of regions of interest means (i) the adoption of a hierarchical data structure and the exploitation of interscale properties; (ii) the definition of common representations of local and global image features and of fine-to-coarse algorithms to allow their quick generation; (iii) a computational platform with coarse-to-fine strategies to locate and redirect attention in space (focusing) and in time (tracking); (iv) the definition of fast high-level mechanisms to implement hypothesis–test cycles.

The special-purpose systems used for the effective implementation of all these processes are the arguments of this book.

2.1.1. Recognition Cones

One of the most impressive human capacities is the great speed of visual perception despite the very slow and simple basic processor, the neurone. In fact, primates perceive complex patterns like faces in 70–200 msec, and this is likely[5] to involve $10–10^2$ serial processing steps (each neurone requires at least 1–2 msec to fire and to distribute its single datum to the interconnected ones). The hardware of the human brain consists of $10^{10}–10^{12}$ neurones, a great portion of which is involved in vision, each one communicating by means of $10^2–10^3$ connections.

The human camera eyes are composed of approximatively 250 million receptor cells each sensible to a portion of the electromagnetic spectrum between 0.4 and 0.7 μm. Each retina contains 6–10 million cones and about 120 million rods with a nonuniform distribution as will be described in Section 2.1.2.1. The rod cells are approximatively 500 times more sensitive to light

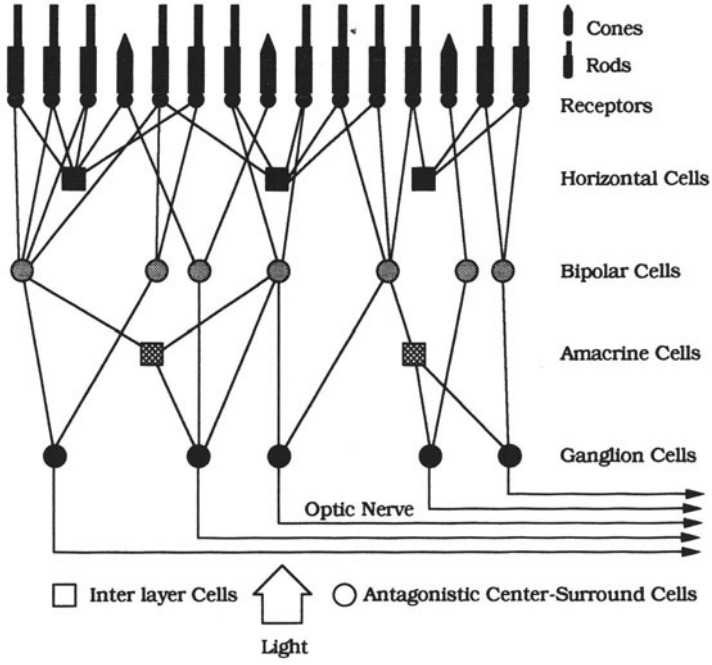


Figure 2.1. Schematic diagram of the retina layers outside the fovea and their interconnection. The two different receptor nuclei are connected to a horizontal layer and to the bipolar, "vertical," one. A new interlevel layer is composed of amacrine cells between the bipolar and ganglion cell layers.

intensity than cone cells, which in turn are differentially sensitive to red, green, and blue components. Indeed a 10^{-14} portion of the light produced by a single candle is sufficient to stimulate the rods.[1]

The retinal pathways between the receptors and the optic nerve follow the sequence of connections illustrated schematically in Figure 2.1. The "vertical"[6] pathway made by the receptor–bipolar–ganglion cells is integrated with a two-level "horizontal" organization consisting respectively of the horizontal and amacrine cells. The first level, which corresponds to the receptor and bipolar cells, contains the horizontal cells. The second level is composed of the amacrine cells and creates a similar horizontal interconnection at the bipolar–ganglion cell layers. These horizontal levels allow the implementation of local and regional manipulations. The retinal ganglion cells communicate through the optic nerve with the inner brain neurones.

The final receptive field of a ganglion cell (area at the retina which influences the behavior of the cell) varies a great deal with respect to the corresponding position in the retina. In particular, the farther they are from the fovea (the area of the retina having the highest concentration of cones), the larger the receptive field of these cells usually is. In the fovea, where the connections between receptor and bipolar cells, and between bipolar and ganglion cells have very few or no lateral interactions, the cone diameter (and consequently the distance) is approximatively 2.5 μm and gives our minimum visual angle at which two points can be resolved: 0.5 min of arc.[7] Instead, on the edge of the retina the receptive field includes thousands of rods, to which corresponds an acuity of more than 1°. Obviously, for the same reason the sensitivity to light is in the reverse order: the integration of several rods produces the extremely high light sensitivity outside the fovea.

The ratio between the number of receptor, bipolar, and ganglion cells in the fovea is close to 1:1:1, whereas in the periphery it is of hierarchical nature (see Chapter 1). The ratio reported by Geldard[8] in this area between rods, bipolar cells, and ganglion cells is 100:17:1. All the quoted authors point out two vertical layers downstream and the concentration of data with a 100:150 ratio between the receptors and the fibers of the optical nerve (ganglion cells). Both bipolar and ganglion cells show excitatory and inhibitory synapses, and they are customarily viewed as contrast detectors. Moreover, half of the ganglion cells are center-surround antagonistic and can be broadly classified as sensitive to ''on-center–off-surround'' or ''off-center–on-surround'' spots.[7]

Uhr, in 1972, introduced the term ''recognition cones''[2] on the basis of this structure of the peripheral human vision system that executes a ''shallow hierarchy of massively parallel . . . micromodular processes to examine successively more global aspects of the scene, extracting, abstracting and combining information as needed.''[5]

From the eyes, through the optic nerves (each composed of approximatively one million axons of the retinal ganglion cells), via one layer of synapses in the lateral geniculate bodies, the visual information arrives at the primary visual cortex. The role of the lateral geniculate body does not seem to be that of a simple relay station. In fact, even if the number of fibers leaving this structure is approximatively equal to the number coming in, the bandwidth is about half; this suggests that a spatial and temporal integration is accomplished.[9]

The primary visual cortex is usually called the striate cortex because of its layered structure. It is located in the occipital lobes. In the 2-mm cortex depth six uniform layers of densely linked neurones have so far been identified. Different layers are communicating with other parts of the cortex, with the lateral

geniculate body, and with the superior colliculus (see Section 2.1.2.1). The activity of most of the cells involves several layers. This indicates, besides layered organization, a columnar one as well, which Hubel and Wiesel have shown to be functionally specialized. In fact, the striate cortex is subdivided into elementary functional units (called *hypercolumns*) with the axis approximatively perpendicular to the surface of the cortex and a section of approximately 1 mm^2.

Each unit is subdivided into two halves, with the dominance respectively from the left eye and the right eye. Each hypercolumn performs the analysis of a precise section of the retinal image (which is called the *hyperfield*). In fact, although strongly distorted, the components of a scene can be topographical localized in the striate cortex (some details on this will be given later). Broadly speaking, the hypercolumns make up a set of coprocessors which perform local feature detection, such as detecting the orientation and motion of simple patterns, and estimating punctual depth.

Obviously, the primary visual area is not the only one involved in vision. It seems to be just a set of local coprocessors which exchange information with at least 20 other secondary visual areas in the cortex. Each of these areas communicates with some of the others through bidirectional pathways. The overall architecture is unknown, but it has been found that each area is linked at most to eight others, and on average to two or three (this conclusion is reported by Uhr[5] on the basis of the physiological and anatomical studies of Van Essen.)[10] Obviously, some of these areas are also involved in nonvisual tasks and may participate in sensory data integration and sensory motor coordination.

Many authors have pointed out that the time required for object recognition depends on object complexity: starting from a few tens of milliseconds for simple stimuli, like bars, and reaching several hundred milliseconds for very complex objects. The ''serial depth of processing'' described by Uhr[5] on the basis of the physiological and anatomical studies by Mishkin *et al.*,[11] includes the sequence of manipulations in the retina, the lateral geniculate body, the primary cortex, and, from here onward, through the two main parallel pathways located respectively in the temporal and frontal lobes. The former path is concerned with the analysis of shape and color, whereas the latter analyzes motion and spatial relationships. These analyses are of course integrated with each other and with the nonvisual perception clues: among the areas involved there may be cycles and feedback loops.

While flowing through the human visual system, scene information, originally coded by scene points in the retina and represented by simple pattern primitives in the striate cortex output, gathers into more and more complex

structures. As this information is processed from peripheral neurones to inner cortical areas, the level of data abstraction gets higher and higher and precision in pattern location becomes smaller and smaller. In other words there is a semantic evolution from "where" toward "what" in the scene.[12]

High-level neurones fire when the stimulus in the scene is a specific one; it seems that there are specializations in "a complex network of information-transform neurones, probably part of a larger structure that fires at that object."[5] However, it is worth noting that there is no single "grandmother cell" specialized for any complex object, and that the computational structure, not being committed to the hardware structure, is flexible.

Following the above considerations, it is possible to broadly quantify the timing of the successive phases in a recognition task. The peripheral processes are clearly massively parallel right from the retina manipulations until they reach the local operations in the striate cortex. According to Uhr[5] they include two to three transformations at the retina level, one in the lateral geniculate body, and from three to six in the primary visual cortex. The total number in the preliminary parallel phase is 6–10 operations. The double parallel path which follows in the secondary visual area consists of a serial sequence that has been estimated as four to six transformations inside each area. The integration and convergence require a sequence of three to six exchanges between the areas, giving a total of 18–46 transformations. If possible loops in the cortical area are included, this total gives rise to the 70–200 msec recognition time.

Concluding this description of human visual hardware, let us point out once more the massive and hierarchical parallelism of the peripheral phase and of the specialized feature detectors (striate cortex) and the mainly sequential processing with exchanges between the different areas inside the cortex (in which a hierarchy can be recognized only for data abstraction). A second general comment refers to the extreme flexibility of the system, not only in selecting paths, loops, and exchanges for the management of cortical information, but even in the capacity for hardware and behavioral adjustments on a training basis (neural plasticity).

2.1.2. Alerting Mechanisms and Peripheral Guidance

According to Marr,[13] vision can be defined as "the process of discovering from images what is present in the world and where it is." The sensing resources of the human eyes are densely concentrated in the fovea, where information gathering is an active, dynamic process. As mentioned, the fovea occupies only a small portion of the field of view of the human eye[1]: the foveal

region with uniform cone distribution at the maximum density is only ± 20 min of arc around the visual axis; the rod-free foveal region is ± 1°–2°; the angle at which the rod density is at maximum is ± 15°–20°; finally, the sensing field is extended broadly over ± 80 ° (see Figure 2.2).

The efficient exploitation of the rich sensing resources of the fovea on such a wide field of view is obtained by sophisticated ''alerting mechanisms,'' which allow us to employ the fovea just in selected areas of the scene and to rapidly guide the focused area through the scene or/and through the images as the task at hand evolves. The areas of inspection of the scene are selected by scanning and analyzing the world around us by eye vergence and head and eye movements. The current eye image is then analyzed by serial shift of ''focal attention,'' which can take place rapidly even without eye movements. Julesz[14] showed how ''focal attention shifts'' are about five times faster than eye movements: attention allocation is driven by eye movements to some ''gross *center of gravity*''; then a detailed scrutiny is performed sequentially on an element-by-element basis by ''a fast-moving aperture of the *searchlight* of attention.''

The astonishing performance in object recognition, using such limited resources, is achieved because of the capability to achieve a *smart* selection of the regions of scrutiny in space, resolution, depth, and time. This strategy will be investigated in some detail in the following three subsections, which cover different aspects: the foveation process toward the area of scrutiny of an image; the tracking mechanism that permits us to follow a target in image sequences; the general framework of control which is necessary to implement these approaches and to establish the sequence of the activities.

2.1.2.1. Foveal Vision

The basic functions of the oculomotor system are (i) to control vision in such a way that the target (scrutiny area) drops into the fovea and (ii) to keep the target into the fovea even if the target or the observer are moving. In 1903, Dodge[15] subdivided the eye movements into two categories: smooth pursuit movements and saccadic movements. In 1961, Rashbass[16] showed that there are two different control systems for these two conjugate movements (some comments on the movements of the eyes in opposite directions, as required for example for eye vergence, will be given later). The former is responsible for slow movements, and its primary function is to move the eyes in a synchronous manner along with the target so that its projection remains stationary on the retina. The latter, instead, is involved in high-speed target movements and in changing the fixation points.

The simplest way to cause a saccadic movement is to suddenly present a new target on the periphery of the field of view. To produce a saccade a deviation of 0.3° from the fovea is sufficient.[17] The speed of eye rotation increases with the amplitude of the eye movement and can reach a maximum of about 600°/sec,[17] in man, for rotations of 30°–40°. Finally, if the required deviation is greater than 15°, the target is often reached by a sequence of a few, usually two, discrete steps (saccades).

There is evidence[6, 9] that the function of guiding the alerting mechanisms and of coordinating the oculomotor activity is done in the subcortical area called midbrain: precisely in the superior colliculus (the two superior colliculi, one for each hemisphere, are connected with the retinas and with the striate cortex). In this center, new presences in the scene are detected and located. Then, on the basis of their eccentricity with respect to the fovea, the saccadic movements, which allow us to foveate and examine the new presences at the cortex's full capacity of analysis, are produced. With these duties, it is easy to understand why the superior colliculus is the most important visual center in the lower animals of the evolutionary scale, such as fishes and amphibians.

As mentioned in the previous section, the alerting analysis is accomplished in parallel over all the field of view; obviously, it is followed by a sequential analysis of the detected scrutiny areas, realized by means of ocular movements. The first phase is often called the *preattentive phase* or early vision phase. The second, detailed analysis of the focused areas is called the *attentive phase*.

Evidence that discrimination of the areas of scrutiny is done in parallel, whereas detection is serial, has been provided by Sagi and Julesz.[18] In different experiments, they showed that the processing time is almost independent of the number of targets (here, object models) to be detected, while it depends on the number of instances present in the visual field to be analyzed. According to the authors, and to our comments in Section 2.1.1, *where* is parallel, *what* is sequential.

The foveation process has been suggested by many authors as a solution to attentive object recognition problems. Figure 2.2 gives details on the nonuniform distribution of receptors in the retina. As pointed out above, visual acuity rapidly decreases from the fovea to the retinal periphery[19]: the angle which corresponds to a 50% drop in visual acuity is only ± 5° around the visual axis. Many authors[20, 21] point out that this acuity distribution (often approximated with the function $1/\theta$, θ being the angle with reference to the visual axis) supports a scale-invariant sensing that can be exploited for object recognition purposes.

A movement in depth relative to the observer makes the retinal image of the target change its size. The monotonic distribution of retinal acuity balances

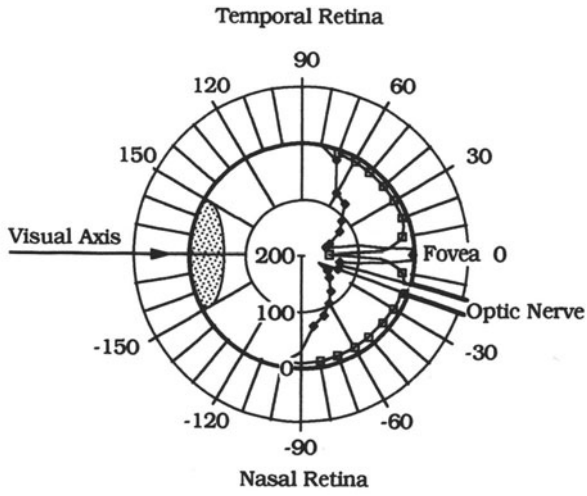


Figure 2.2. Distribution of the receptors in the human retina. Diamonds (♦) represent rods, squares (□) represent cones. The height of the curves along the normal to the circle represents the density of receptors in millions/mm^2. In the nasal area at the exit of the optic nerve the absence of receptors produces a "blind spot."

this effect with an opposite variation in acuity. In this way, by means of appropriate eye movement, details can be made invariant with respect to the viewing distance. Consequently, visual hardware can support scale-invariant object recognition strategies.

In conclusion, the nonuniform distribution of the acuity, combined with dynamic control of eye movements, provides a very effective mechanism which supports event selection and perceives invariant primitives for recognition purposes.

2.1.2.2. Locking into the Target

The main objective of the second conjugate oculomotor system, the "smooth pursuit," is to stabilize the target image on the retina. In order to achieve this, the eyes must be moved with a speed correlated with the target speed. Saccadic movements mainly depend on target eccentricity with respect to the fovea. Instead, smooth pursuit movements mainly depend on target velocity, and their control system can be considered basically as a velocity servo. Due to static and dynamic limitations, tracking of fast targets may require con-

tributions from saccadic and smooth pursuit systems. In appropriate conditions, smooth pursuit velocity can reach about 100°/sec.[22]

In fact, target tracking requires another control system for vergence, which allows us to follow the target movements in depth with respect to the observer. In this case, eye movements are the opposite. More precisely they must converge if the target is coming closer to the observer, and diverge in the opposite case. There are three other eye movement control systems, the vestibulo-ocular reflex (VOR), the optokinetic reflex (OKR), and a reflex originating from the neck: proprioception. They compensate retinal image movements due to observer's movements in a stable visual environment and are briefly described here.

The role of foveation in space is accomplished by tracking in time. In fact, tracking allows a selective analysis by isolating regions in successive time (frame in computer vision), thus simplifying target motion analysis in a dual way with respect to the selection in space of the regions for the detailed analysis.

Human tracking is simplified by the VOR and the OKR. They are tightly coupled in one functional unit which philogenetically represents the oldest oculomotor control system. Its role is to suppress the scene motion components due to head movements in space. The reflex originating from neck proprioception (cervico-ocular reflex, COR) is less well known. It is assumed that its role is to compensate scene motion due to head rotation with respect to the body.

VOR and COR exploit neural signals coming from receptors of the vestibular system of the inner ear and from the proprioceptors, respectively, as input signals for compensation. They basically work in an open loop. The loop is closed by OKR, which is stimulated by the residual image slip on the retina (*retina slip*) and acts to suppress it.

The three reflexes converge and interact at the level of the vestibular nuclei of the brainstem. Coordination is obtained thanks to the complementary performance of each reflex and, in particular, to the negative-feedback, closed-loop structure of OKR.[23, 24] This process is supervised by the cerebellum.[25] The vestibular nuclei receive head motion information of vestibular (i.e., inertial), proprioceptive (i.e., mechanical), and optokinetic (i.e., visual) origin and can then reconstruct the absolute head movement. This is used for ego-motion and object-motion sensations.[23, 24, 26]

The human tracking system is so effective that even small objects slowly moving relative to the tracked and stabilized background stand out: the minimum speed to detect a target in the fovea is 1–2 min arc/sec, and progressively increases going toward the periphery of the retina.[9]

2.1.2.3. Where to Look Next

The only way to effectively implement analysis of the current scene is to be able to predict, or at least to easily detect, where the salient information to accomplish the visual task at hand is likely to occur. This can be achieved in two ways: the first is by having at least a partial understanding of the content of the scene; the second is to be able to rapidly locate the key items for analysis.

The analysis of the order of eye fixation points, or *scan path* following some authors, has been deeply investigated, starting from the well-known experiments by Yarbus[27] (see Figure 2.3). As can be easily derived from inspection of the bust of the Egyptian queen Nefertiti, the scan path is by no means random, but the ''salient'' features of the scene (mouth, nose, eyes, ear) are inspected following a dense and ''uniform'' pathway. This regular sequence of the salient features has been called *feature ring*.[28] Noton and Stark suggested, in the paper quoted, that the feature ring is used for remembering and recogniz-

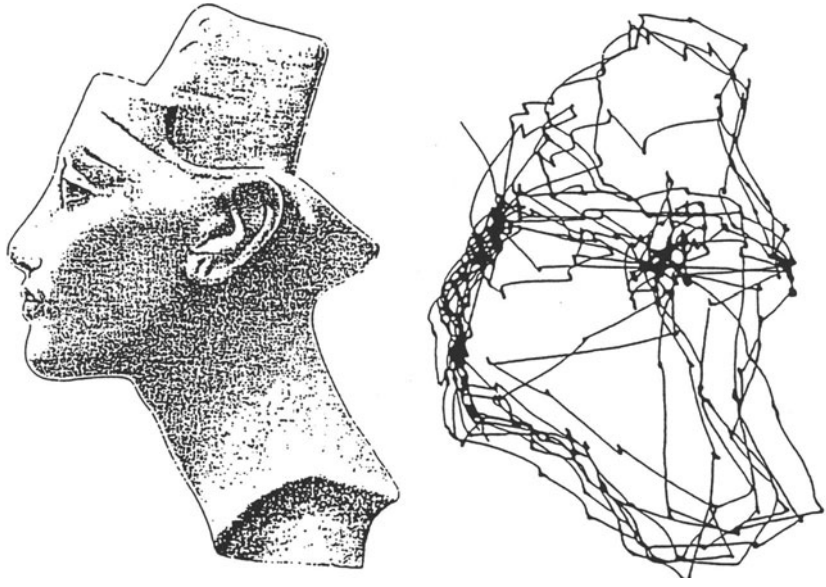


Figure 2.3. Eye fixation pathway generated when looking at the Egyptian queen Nefertiti. Note the regular sequence of ''critical'' points, each one representing significant features for image evaluation. (Courtesy of A. L. Yarbus, *Eyes Movements and Vision,* New York, Plenum Press, 1967.)

ing objects. In particular, the feature ring could be used as a test verification strategy defined *a priori*. Each object can be represented in memory by the feature ring, which becomes the model, and the sequence of eye movements for recognition. This hypothesis is controversial; however, like many other conjectures concerning human object recognition strategy, it is based on the assumption that recognition implies an active comparison between the salient features of the object detected and those of the object models stored in the brain as a result of previous experience.

The cited experiment, even if it does not prove (and probably this is not the case) that the human-eye scan path is always the same for a given object, shows that the scanning sequence takes care of a few important features. Julesz claims that scan path is one of the most misused words by vision scientists.[14] Certainly, the features usually considered in feature rings correspond to the contour critical points described by Freeman[29]: the points with highest curvature and the angles. These are commonly used for recognition purposes and are also considered for object synthesis to allow shape recognition with a compact description.

On these points, the scrutiny of the unknown object can be based. Starting from the most peculiar ones, recursively, as new information is gathered, new hypotheses are formulated, the system directs eyes toward the missing details in order to confirm or reject the hypotheses until there is sufficient evidence for recognition. During these phases of hypothesis and testing, complete integration of cortical and peripheral visual analysis occurs.

According to Burt,[21] this recursive interplay between peripheral sensing and interpretation is the key difference between traditional image processing systems and the human vision system. To implement this approach in computer vision systems, it is ncecessary to speed up the high-level reasoning capability in order to generate the strategy and the control for the selection of the areas of interest (*where to look next*) in real time. Because of the enormous amount of data generated with standard visual sensors, this cannot be achieved by working directly with the data generated uniformly at high resolution.

2.2 ALLOCATION OF ATTENTION IN COMPUTER VISION SYSTEMS

Traditional image processing systems are characterized by the implementation of a fixed cascade of steps: the image is collected, preprocessed, segmented, analyzed, and interpreted or classified. This static computational para-

digm mismatches the strict interaction needed to execute in real time even the simplest task that human beings normally execute without apparent effort.

Raw visual data emerge from the standard transducing sensors at the rate of 10^6–10^7 pixels per second, which are uniformly distributed on the camera field of view. Most of the information collected is unproductive, and useful data have to be extracted from this huge amount of ineffective data. Conventional computers are unable to deal with this selection problem. It is necessary to specialize the computer vision systems to identify the regions and the events of interest in a manner equivalent to that of the higher-order animal vision systems.

The human attention mechanism operates on the same huge amount of data by redirecting its sensing resources to just the data necessary to accomplish the tasks needed, switching in a predefined, shortly determined sequence between the salient areas to allow scrutiny. Analogously, a computer vision system must be capable of (i) rapidly selecting the regions of interest in space and time; (ii) reallocating computer resources, maintaining efficiency even if the data load changes considerably; (iii) changing, as required, the data resolution, in order to operate with local and global (or regional) features; (iv) rapidly updating the strategies on the basis of the partial results achieved step by step.

The most promising proposals that directly address the cited requirements are based on multiresolution processing.[30–32] This approach consists of analyzing the images through a pyramidal data structure. This data structure supplies the same image at different resolution levels (see Figure 2.4), thus ensuring the use of the most appropriate resolution for the operation, task, and image at hand. As a general strategy, large components can be analyzed at low resolution, while small components are handled at the highly detailed levels. Nevertheless, independently of the target size, a considerable reduction in the amount of data to work with is gained every time the property involved is scale independent. Furthermore, pyramid data structures allow (i) easy detection of local and regional features by selecting the appropriate scale; (ii) coarse-to-fine detection and analysis strategies; (iii) detection of interscale properties.

In the literature it is possible to find other data representation approaches that have a hierarchical structure,[33, 34] among which are the quad-tree data structure, which is based on the description of a single-scale image by a variable-resolution hierarchical structure,[35] and other representations adopting a hierarchical recursive grouping of features based on a single-scale resolution image.[36, 37]

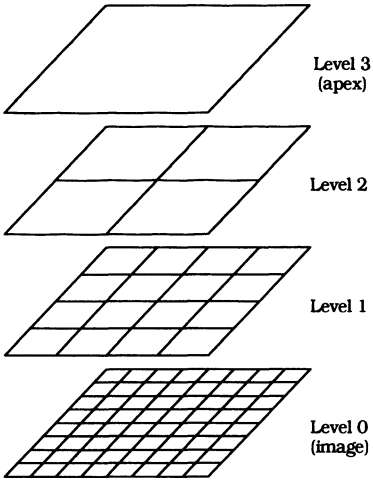


Figure 2.4. The multigrid data structure resulting in a quad-pyramid representation. The size of the tessellation varies with power of 2 steps in the linear image dimension.

2.2.1. Multiresolution Model Representations

In the last decade three different multiresolution model representations were introduced which allow us to integrate the hypothesis-test paradigm for the recognition process. Moreover, such representations may be considered as guidelines for emulating the focus of attention typical of biological systems.

The first two multiresolution models were similar. One is called a *pattern tree,*[21] in which subpatterns (having significant relevance for the overall recognition process) are represented as nodes of a tree (see Sections 3.5.1 to 3.5.3), each descendent node containing a fraction (at a higher resolution level) of the subpattern present in its parent node. Instead, the second model, called *model feature graph,*[32] is an acyclic graph where nodes, corresponding to subpatterns or features, are organized into levels. General nodes at a given level may be reached via different paths from the previous levels.

A third general method for 2-D object description is based on a linguistic approach,[38] where the object is described on each level by means of a context-sensitive grammar: in practice the image variations from a coarse to a finer level are coded by production rules.

In the following these three proposals will be discussed in some detail.

2.2.1.1. Graph-Based Object Representations

In this section we review two frameworks for representing pattern information over a wide variety of scales: the pattern tree and the model feature

graph. These representations support fast coarse-to-fine search procedures for dynamic vision analysis and object recognition.

In a pattern tree, subpatterns (having significant relevance for the overall recognition process) are organized along nodes of the tree, with each descendent node containing a fraction (at a higher resolution level) of the subpattern present in its parent node. A full pyramid of images, taken at various levels of resolution, is first constructed (for instance, a Gaussian pyramid,[39] see Section 2.4.2) and then subarrays are selected in each image representing object subpatterns. The size of the subarrays is roughly constant throughout the pyramid so that progressively smaller subpatterns are represented as moving away from the root node. Subarrays considered at a given level of the pyramid are not necessarily disjoint since distinctive subpatterns may overlap.

The search for a given object is performed through a sequence of simple matching steps; the results of each step guide the search at the next step. The links in the tree help to locate the subpatterns wanted.

If an object is partially occluded, it is still possible to identify it provided that a distinctive subpattern, represented by a node in the tree, is visible. Nevertheless, for an unambiguous representation of occluded or noisy objects, the next data structure is more adequate.

The second data structure, the model feature graph (MFG), is a directed acyclic graph where nodes representing local features or subpatterns are organized into levels corresponding to different resolutions, and nodes at a given level may be reached via different paths from the previous level. This is particularly useful in cases of occlusion: if a feature represented by a parent node is occluded but there are other parent nodes, there still remains a possible path that allows us to exploit the feature of the current node.

An interesting aspect of this representation is the specification of the match set,[40] that is, a subset of nodes of the graph belonging to a connected path that allows us to discriminate between the given object and all other possible objects. To this end, each node is associated with a weight (smaller than 1) specifying the significance of the feature allocated to that node. The weights are selected in such a way that any connected path with a total weight equal or greater than 1, unambiguously identifies the object. A match set strongly reduces the search time, since it by itself provides evidence of the presence of the object in the observed image.

In the model feature graph presented in the cited paper, the features represented in the graph nodes are sets of unconnected edge points stored in a generalized Hough transform R-table.[41] The matching processes performed at each level of the search are based on the voting results of the generalized Hough transform.

2.2.1.2. The Syntactic Description

The well-known structural approach to pattern recognition[42, 43] represents patterns as strings where the patterns are considered at a single resolution level and the decision-making process for recognition is based on a parsing procedure of the string obtained at the first stage of the analysis. Each object class is represented by a formal grammar which accepts the string representing the object to be recognized. The unknown pattern is coded in a sentence which will be parsed, providing a yes/no answer according to whether the object belongs to that class or not. Each grammar may be defined as a quadruple $G=(V_n,V_t,P,S)$, where V_n (V_t) is the symbol vocabulary for nonterminal (terminal) symbols, P represents the production rules, and S is the starting symbol.

The general approach, based on a single-level representation of the object, implies that the V_n symbols do not always have a direct physical counterpart. Instead, in the multiresolution method,[38] based on patterns at different resolution levels, there is no difference between terminal and non-terminal symbols: they all correspond to some specified topological properties of a boundary segment. Patterns are described at each resolution by means of the same set of terminal symbols.

Production rules formalize the process through which detail is augmented as the resolution is increased: each single production rule refers to the same subpattern at the two consecutive levels between which salient feature(s) emerge. More specifically, the production rule describes the subpattern evolution between the low- and high-level resolutions. In this multiresolution grammar the start symbol S corresponds to the first coarse representation of the object, which can be seen as a blob.

2.2.1.3. An Example

Figure 2.5a illustrates the silhouette of a fish represented at six different resolution levels by the so-called Gaussian pyramid. Each level is binarized, as can be seen in Figure 2.5b, to be later labeled in terms of the topological properties of the contour. In this example, five different classes of ''curvature'' have been defined, thus obtaining the corresponding Figure 2.5c.

For each different resolution level, a string may be constructed after associating each topological property with a symbol as shown in Table 2.1. By comparing the generated strings at the different consecutive levels, production rules describing contour evolution may be extracted. Table 2.2 reports, for each resolution level, both the contour descriptive string and the corresponding production rules used.

Table 2.1. The set of terminal (and nonterminal) symbols adopted in the grammar of Table 2.2 to describe the multiresolution model of the fish in Figure 2.4.

Terminal symbols V_t	Labels
Very concave	W
Concave	C
Straight	S
Convex	X
Very convex	Y

An overview of the labeling process can be seen on the graph in Figure 2.6 which corresponds to the model feature graph (and to a possible pattern tree). In fact, the linguistic and the acyclic graph representations are isomorphic and differ only in the way they code and handle the pattern information: for a different ordering of the rules a corresponding MFG exists. For the example in Figure 2.6 representing the rules of Table 2.2, top to bottom and left to right precedences were chosen.

Table 2.2. The first column indicates the resolution levels, the second the strings describing (following Table 2.1) the silhouette of the object, and the third shows the production rules when a new detail appears

L	Starting string	Production rules
5	$X^1, S^1, C^1, X^2, C^2, X^3, S^2$	$\rightarrow X^1, S^1, C^1, X^2, C^2, X^3, S^2$
4	$X^1, S^1, C^1, X^2, C^2, X^3, S^2$	
3	$X^1, S^1, X^2, S^2, X^3, C^1, X^4, S^3, X^5, C^2, X^6, C^3, S^4$	$S^1 \rightarrow S^1, X^2, S^2, X^3, X^2 \rightarrow X^4, S^3, X^5, S^2 \rightarrow C^3, S^4$
2	$X^1, S^1, C^1, X^2, C^2, X^3, C^3, X^4, S^2, X^5, C^4, X^6, C^5, S^3$	$S^1 \rightarrow S^1, C^1, S^2 \rightarrow C^2$
1	$X^1, S^1, C^1, X^2, C^2, X^3, C^3, X^4, C^4, X^5, C^5, X^6, C^6, S^2$	$S^2 \rightarrow C^4$
0	$Y^1, S^1, C^1, Y^2, C^2, Y^3, C^3, Y^4, C^4, Y^5, C^5, S^2, C^6, Y^6, C^7, S^3$	$X^* \rightarrow Y^*, C^5 \rightarrow C^5, S^2, C^6$

Note that each symbol instance is labeled according to the order it appears in the string, and the implicant label refers to the level above, while the implied label is the one pertaining to the current level If a label (new detail) is originated from two adjacent subpatterns, it will appear in both consequent production rules

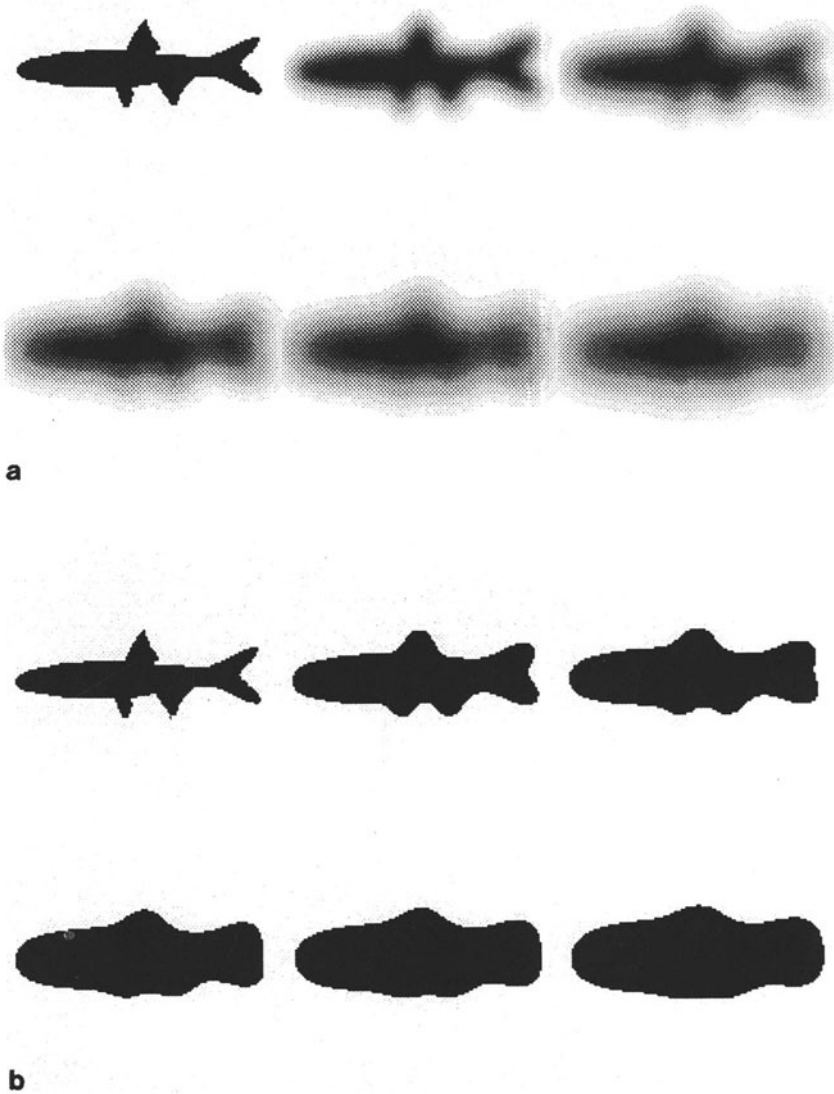


Figure 2.5. (a) Six levels of a Gaussian pyramid representing a fish expanded to a standard common size; (b) thresholded version of (a); (c) labeled silhouette of the content of (b) after contour segmentation (see Table 4.1 for symbol meanings).

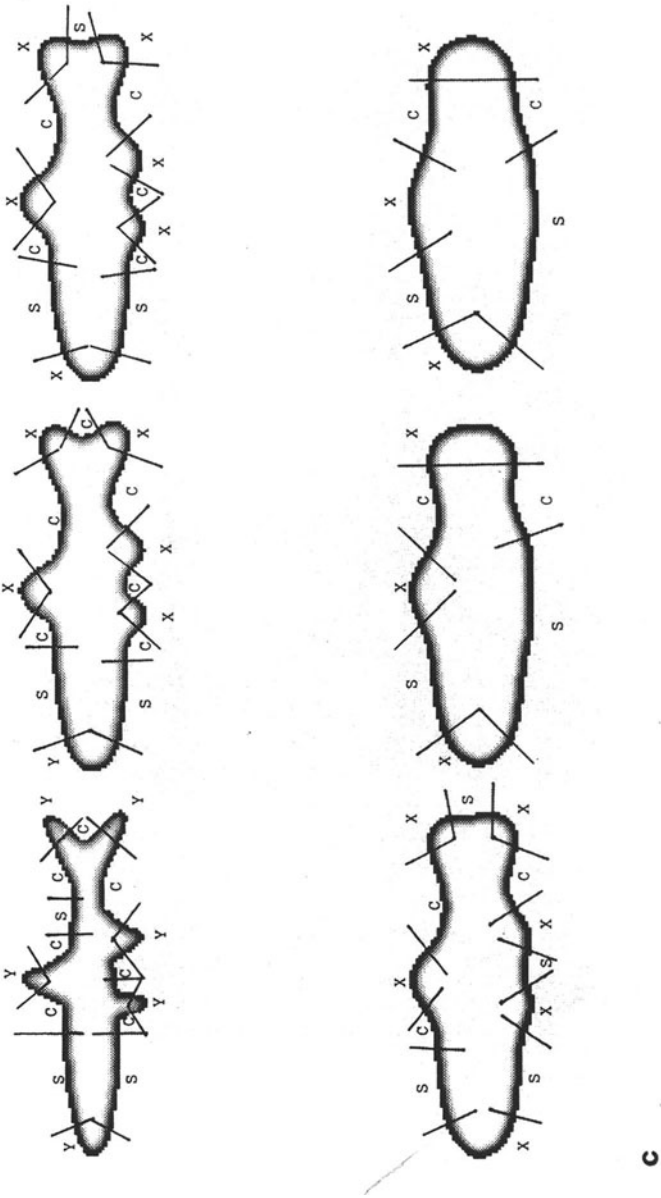


Figure 2.5. *Continued.*

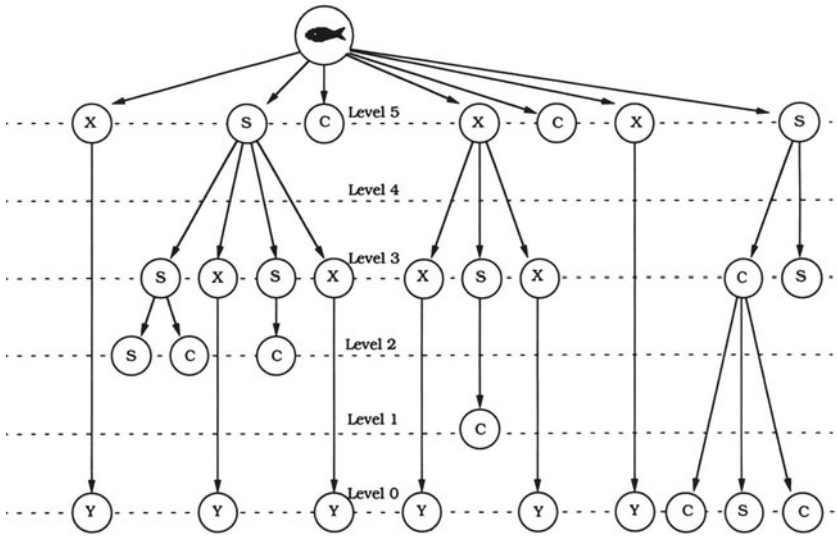


Figure 2.6. Model feature graph corresponding to the production rules of Table 2.2. The full strings can be easily obtained by replicating leaf nodes downward to the bottom.

2.2.1.4. Recognizing Objects

These representations turn out to be an explicit description of the mechanism for directing attention in the object recognition process. In addition to a description of the object at different resolution levels, these representations support the cues for the next focus of attention. The global recognition process may be subdivided in this way by a step by step series of elementary partial matches in which each match (mismatch) guides the successive step in the process. This mechanism models the eye movements performed in humans when observing objects to be recognized, as described in Section 2.1.2.

The basic strategy of this approach is to start from a coarse version of the object and, gradually, in a guided manner, reach the finer resolution levels where the significant details may be found. Three important issues must be considered for a correct implementation of this strategy:

- The coarsest level should be chosen so as to reliably establish the branch to be expanded from this node downward.
- The finest level should be enough to discriminate the object to be recognized among all possible objects that may be present in the scene.

- The possibility of verifying a correct recognition is available on all levels of the tree: for levels above the basic one by simply going in coarse-to-fine mode and for the basic level by observing the isolevel details.

As for all problems where the solution has to be found by navigating through the hierarchical graph (see, for instance, Ref. 44) backtracking can be used, besides the depth-first node expansion, as well as width-first and many other parallel solutions.

In a recognition process under the constraint of a completely known closed world the optimal strategy is easily obtained on the basis of all the models of the possible objects of the scene. To this end, the training techniques based on learning by example (e.g., the ones based on the detection of the *minimal discriminant description* and the *maximal conjunctive generalization*[45]) may be used. Furthermore, in order to increase the recognition rate and to enable features attribute weighting of the objects to be recognized, a possibility to be considered is the extension of the production rule method to attributed grammars[46] so as to include metrics in the recognition process.

2.3. MULTIRESOLUTION MATCHING

One of the most challenging aspects of vision is that often processes must be size independent: ordinary scenes include objects of many sizes, with many-sized features, at an arbitrary distance from the sensor. Matching processes under these arbitrary size conditions are very time consuming. In fact, the execution of the convolutions with kernels of different sizes requires a sequence of operations of the order of $O(N^2T^2)$, where as usual N is the linear size of the image and T is the linear size of the target.

Even if the search for a target which can appear everywhere in the image can be more effectively performed with different approaches, let us first show quantitatively the advantages of a direct multiresolution match (similar quantitative analysis can be found in Burt[3] and Dyer[32]). Let us consider the case of a target that can appear with s different scales, with a ratio M between two consecutive scales. The number of operations O_f in image space of fixed size with the various target scales will be

$$O_f = O\left(N^2\left\{T^2 + \frac{T^2}{M^2} + \cdots + \frac{T^2}{M^{2(s-1)}}\right\}\right) = O\left(N^2T^2\frac{1-M^{-2s}}{1-M^{-2}}\right) \qquad (2.1)$$

In a multiresolution environment (that is, supposing that we have the image at different sizes, with module M: N^2; N^2M^{-2}; N^2M^{-4}; . . .) the detection

of targets of different sizes can be executed by correlating the various image scales with a fixed model target. More precisely, the search for the most scaled-down target is executed on the highest-resolution image. Larger targets are identified by correlating smaller and smaller images with the same scale-down instance of the target. Obviously the effect will be the same as correlating the standard image with enlarged targets. The number O_m of operations in multiresolution image space with the fixed target space will be

$$O_m = O\left(\frac{T^2}{M^{2(s-1)}}\left\{N^2 + \frac{N^2}{M^2} + \cdots + \frac{N^2}{M^{2(s-1)}}\right\}\right) = O\left(\frac{N^2T^2}{M^{2(s-1)}}\,\frac{1-M^{-2s}}{1-M^{-2}}\right) \tag{2.2}$$

The resulting ration R between the two costs of correlation is

$$R = \frac{O_f}{O_m} = M^{2(s-1)} \tag{2.3}$$

Under the assumption that the resolution of three levels for both images and targets is sufficient to allow identification (e.g., $N=512$, 256, 128, $T=64$, 32, 16, and with the usual quad-tree pyramid having $M=2$): the speedup coming out of Eq. (2.3) is 16.

It is well known, that direct correlation is not the general solution for object recognition by matching. Nevertheless, several more effective solutions are based on pattern matching by correlation, e.g., a sequence of steps which include correlation at low resolution and then verification and refinement of the solutions at the finest levels. The coarse-to-fine analysis limited to the regions of interest does not change the conclusion of the analysis above (we will see some details of it in Section 2.6), which shows that the multiresolution approach permits us to gain at least one order of magnitude in speedup.

Correlation is not the only image processing task that benefits from multiresolution; many object properties that are scale independent (e.g., edges, shapes, and textures) allow us to reach similar performances (see Chapter 10).

2.4. FINE-TO-COARSE FEATURE GENERATION

In a pyramid data structure, images are represented by a sequence of copies of the original data in which both sample density and resolution are de-

creased in regular steps, from the base (containing the original image) toward the apex. Some irregular solutions are analyzed in Chapter 10. Figure 2.4 shows a uniform square tessellation; this is not the only possible solution, and a general analysis of the possible tessellation suitable for hierarchical composition is given by Tanimoto.[47] In the following we will consider the square case, by far the most commonly used, and the second most important hierarchical solution, hexagonal tesselation. A detailed analysis of this last case can be found in Burt,[48] Ahuja,[49] and Hartman and Tanimoto.[50]

The degree of reduction between successive image copies (levels subsequently) can vary over a wide range. In a common solution the number of pixels is halved at each successive level. Some authors[51, 52] in an isotropic way, but adopting different approaches, use a scaling factor $\sqrt{2}$ for each linear dimension. Others generate the so-called bin pyramid by resampling alternatively in either direction, 1×2 and 2×1, as described in detail in Sections 3.5.4 and 5.2.4.4. In a quad pyramid the scaling factor is 4 (2 for each linear dimension). Smaller ratios have been proposed till the limit case considered by Uhr was reached.[5] Here the convergence is not logarithmic but linear: a constant (e.g., unitary) decrement is applied for each dimension. In the following parts of this chapter, the structure discussed will be the quad pyramid; but remember that all results can be easily extended to the other regular cases mentioned.

In a pyramid data structure, images are represented in multiresolution and, going toward the apex, the bandwidth of the various levels decreases. In fact on the basis of the Nyquist limit, only low-spatial-frequency components can make up the data in the higher layers of Figure 2.4. In particular, in the quad-pyramid case, the bandwidth decreases uniformly along levels in one octave, or power-of-2, steps.

The pyramid construction from the original image can be accomplished in many ways: the simplest way consists of direct subsampling; each level is obtained by discarding the even rows and columns from the previous level (hardware has been conceived that achieves this result in zero steps just by loading the image from a camera; see Section 7.2.3.1); other, more sophisticated, solutions are usually pursued in order to preserve or to enhance some image properties. In what follows we analyzed the formal definition of a few pyramid representations.

2.4.1. Wavelet Representation

An analytical foundation of the notion of multiresolution, specifically of multiresolution signal representation, was introduced by Mallat.[53] It is based

on the wavelet representation: a set of approximations of a signal (image) is produced at different levels of resolution and is used to build the actual multiresolution representation. This consists of a coarse version of the original signal, plus a set of ''detail signals.'' The detail signals are the differences between successive reduced resolution approximations.

The wavelet decomposition is complete and not redundant: the original image can be reconstructed without distortion, and the number of data in this representation is equal to the number of input samples. These properties are due to the fact that the wavelet decomposition uses an orthonormal basis.

This decomposition is a multiresolution representation which has the characteristic that the functions of the basis are similar at all resolutions. In fact, they are generated from a single function by scaling and translation. Moreover, the discrete approximated versions of the input signals can be computed by convolution with a quadrature mirror filter and a subsequent decimation in space (see Sections 2.4.1.3 and 2.4.1.4).

The representation of bidimensional signals (images are the most common case) can be tuned to be sensitive in any chosen direction. If we consider a separable basis, the simplest wavelet is the set of the Haar basis functions. The Haar transform emphasizes image discontinuities in the two main orthogonal directions (a somewhat detailed analysis of this transform is given in Section 2.4.4). Nonseparable functions can be selected to yield sensitivity in a single specific direction.

The wavelet representation has both theoretical and practical interests. It sets a framework for all nonredundant multiresolution image representations and can therefore be used as a yardstick for other representations. In the following sections, we will discuss some alternative image pyramid representations, e.g., the Gaussian and the Laplacian pyramids. The wavelet decomposition will be used as a framework for studying their characteristics in the context of signal theory.

In regard to applications, the wavelet transform is being used especially in speech analysis, image coding, and progressive image transmission. Some proposals have been put forward to create new standards for HDTV, based on such a representation as a replacement for the more common discrete cosine transform.

In the following, we briefly summarize the definitions of Mallat by analyzing first the monodimensional case and later extending the analysis to the bidimensional one. The section closes with a short review of other orthogonal representations that can be considered as closely related to wavelets.[54]

2.4.1.1. Multiresolution Vector Spaces in $L^2(R)$

Mallat formalizes the notion of multiresolution in the context of finite-energy signals represented as functions in the vector space $L^2(R)$. In an analogous way, the refinements of hierarchical modular systems were introduced in Chapter 1. The normalization is achieved by choosing a set of subspaces V_r, each of which allows us to represent any signal at the corresponding resolution r. According to Mallat, such subspaces are a multiresolution representation of $L^2(R)$ if the following conditions hold:

- The sequence of resolutions is denoted with the values of r such that $r=2^i$, $i \epsilon Z$; i.e., the original signal is defined at resolution 1, coarser approximations are defined at resolution $\frac{1}{2}$, $\frac{1}{4}$, $\frac{1}{8}$, and so on.
- $V_r \subset V_{2r}$: each new coarser subspace V_r is already available in the finer subspace V_{2r}.
- The subspaces are dense in $L^2(R)$.
- Let $g(x)$ be a function in V_r. Then $g(2x)$ is a function in V_{2r}; that is, the approximating subspaces are made up with functions that are similar at all resolutions and that can be derived from one another by scaling.
- The self-similarity property holds within each subspace and any function in the subspace can be defined by a set of r samples per unit length: $g(\mathrm{x}) \epsilon V_r \Leftrightarrow g(x-r^{-1}k) \epsilon V_r$, $\forall k \epsilon Z$.

2.4.1.2. The Scaling Function

Given a multiresolution representation V_r of $L^2(R)$, Mallat shows that a single function exists: $\phi(x) \epsilon L^2(R)$, which acts as a generator for an orthonormal basis for each subspace V_r. By defining $\phi_r(x)$ as the dilation of $\phi(x)$ by r: $\phi_r(x)=r\phi(rx)$, the orthonormal basis for V_r is the set $\{r^{-1/2}\phi_r(x-r^{-1}k)$, $\forall k \epsilon Z\}$. $\phi(x)$ is the scaling function of the chosen multiresolution representation of $L^2(R)$.

According to this result, the approximation of any signal $f(x) \epsilon L^2(R)$ at a given resolution r, denoted by $f_r(x)$, can be derived in the reduced resolution subspace V_r as an expansion over the basis $\phi_r(x)$:

$$f_r(x)=r^{-1}\sum_{n=-\infty}^{n=\infty} f_r^d(n)\phi_r(x-r^{-1}n) \tag{2.4}$$

The coefficients f_r^d of the expansion are known as the *discrete approximation* of the signal $f(x)$ at resolution r. The computation of this discrete approxi-

mation f_r^d is indeed quite a simple procedure. Each component of the approximation is the outcome of the convolution of the original signal $f(x)$ at its full resolution with the scaling function $\phi_r(x)$, followed by a subsampling with a rate r. The scaling function acts as the kernel of a low-pass filter. This relationship is made explicit in the following section.

2.4.1.3. The Pyramid Generation of Approximations

The various discrete approximations of the input signal can be obtained recursively and hierarchically from one another through the following procedure.

Let H be the filter having impulse response h given by the convolution of $\phi_1(x)$ with $\phi_{1/2}(x)$, and H_m the mirror filter of H with impulse response $h_m(k) = h(-k)$, $\forall\, k \epsilon Z$. Then the discrete approximation at resolution r can be obtained from the approximation at resolution $2r$ by convolution with the filter H_m and subsampling by a factor of 2. By using the notation introduced by Burt,[60] indicating the convolution operator with (*), it can be stated more concisely as

$$f_r^d = [h_m * f_{2r}^d]_{\downarrow 2} \tag{2.5}$$

where $[\cdot]_{\downarrow 2}$ indicates a subsampling by a factor of 2 of the signal contained between brackets.

The discrete filter H_m is also known as a *generating kernel*. It depends only on the sampling function. The choice of the subspaces V_r, and consequently of the scaling function and generating kernel, discriminates among the possible multiresolution representations. In Section 2.4.2 we consider the case of Gaussian approximations. The peculiarity of the wavelet scaling functions is due to their orthogonality. As a result, the filter $H(\omega)$ is also a *conjugate* filter:

$$\begin{aligned} &|H(0)| = 1; \qquad h(n) = O(n^{-2}) \quad n \to \infty \\ &|H(\omega)|^2 + |H(\omega + \pi)|^2 = 1 \end{aligned} \tag{2.6}$$

which is a sufficient condition for the property of completeness, described in the following.

An extensive study of the properties of the such filters is given by Millard and Paul.[55]

2.4.1.4. The Detail Signal and Wavelet Decomposition

The multiresolution approximations $f_r(x)$ of the original $f(x)$ are obviously a set of incomplete replications of the input signal. The information contained in them is not an alternative way to represent the original data in multiresolution. Wavelet decomposition achieves this result. It consists of a multiresolution representation of $f(x)$ which is complete and not redundant.

Wavelet decomposition is based on the *detail signals*. A detail signal $d_r(x)$ at resolution r consists of the difference between the approximations $f_{2r}(x)$ and $f_r(x)$. Just as the subspace V_r has a basis for the decomposition of $f_r(x)$, so the subspace U_r, which is the orthogonal complement of V_r in V_{2r}, has a basis for the decomposition of $d_r(x)$.

The generating function $\psi(x)$ of these basis is called the *orthogonal wavelet* and is derived by the scaling function $\phi(x)$ through the following relationship in the Fourier domain:

$$\Psi(\omega) = \mathrm{G}\left(\frac{\omega}{2}\right)\Phi\left(\frac{\omega}{2}\right) \tag{2.7}$$

where $\mathrm{G}(\omega) = e^{-i\omega}\,\overline{H(\omega+\pi)}$

The wavelet basis is a set of bandpass functions. The relationship between the wavelet basis and the scaling function basis, as highlighted in (2.7), shows that the filter G is a high-pass filter. G and H together are a *quadrature mirror filter* pair.[55]

The detail signals $d_r(x)$ can be described similarly to the approximations $f_r(x)$ as a set of discrete samples d_r^d obtained by convoluting $f(x)$ with the proper wavelet $\psi_r(x)$, followed by subsampling.

The orthogonal wavelet representation of a signal over L resolution levels is

$$f_{2^{-L}}^d,\ d_{2^{-L}}^d,\ d_{2^{-L+1}}^d,\ \ldots,\ d_{2^{-1}}^d \tag{2.8}$$

The representation includes the approximation $f_{2^{-L}}^d$ at the coarsest approximation chosen and the detail signals on each finer level, up to the next-to-last $d_{2^{-1}}^d$ (1 being the resolution of the original signal).

The discrete signals in this representation contain a number of values equal to the samples of the input signal. This property (nonredundancy of the representation) is due to the fact that the wavelets $\psi_r(x)$ are on an orthonormal basis.

Mallat shows that there exists a multiresolution algorithm to recursively derive the detail signal d_r^d at resolution r from the detail signal d_{2r}^d at the finer

resolution $2r$. Again, the relationship is the convolution with the mirror filter G_m derived from G, followed by subsampling.

The wavelet representation is complete. The original signal can be reconstructed with an iterative procedure. Each finer approximation f^d_{2r} is obtained by the coarser one and by the detail signal: the two discrete series of values are expanded and padded with zeros (denoted by $[\cdot]_{\uparrow 2}$), convoluting the expanded signals respectively with the low-pass and high-pass filters H and G, and adding the outcome of the convolutions:

$$f^d_{2r} = 2h * \left([f^d_r]_{\uparrow 2}\right) + 2g * \left([d^d_r]_{\uparrow 2}\right) \tag{2.9}$$

2.4.1.5. Bidimensional Wavelets for Images

The wavelet transform can be extended to higher-dimensional spaces $L^2(R^n)$, $n > 1$, with straightforward modifications of the monodimensional case. The modifications adapt the definitions of multiresolution vector spaces, scaling functions, approximated signals, associated wavelet functions, and detail signals, along with the pyramid algorithms for the recursive construction of the approximations and for the exact reconstruction of the original signal from wavelet decomposition.

The bidimensional case is especially interesting when the chosen scaling function is separable: $\mathcal{F}(x, y) = \phi(x)\phi(y)$. Mallat shows that in such a case the wavelet basis for $L^2(R^2)$ consists of the following three bidimensional wavelets:

$$W_1(x, y) = \phi(x)\psi(y); \quad W_2(x, y) = \psi(x)\phi(y); \quad W_3(x, y) = \psi(x)\psi(y)$$

Such a ternary basis emphasizes image elements along the cardinal directions.

The bidimensional wavelet decomposition partitions the frequency spectrum into four areas: a low-frequency domain (associated with the coarse representation of the original signal), a domain containing energy in the high frequencies for horizontal transitions (associated with W_2), a domain containing energy in the high frequencies for vertical transitions (associated to W_1), and a domain containing energy in the high-frequency diagonal transitions (associated to W_3).

The pyramidal algorithm used to build the wavelet decomposition from the original signal (coarse approximation plus detail signals) and the expansion algorithm used to reconstruct it are based on the application of the separable conjugate quadrature mirror filters G and H introduced in the monodimensional case. This means that the discrete bidimensional signals are derived by a first convolution along the rows, then by discarding every other column, by a con-

volution along the column, and finally by discarding every other row. When the kernel of the chosen filters is very small, a direct bidimensional pyramid implementation is possible. We will consider this situation in the special case of the Haar transform in Section 2.4.4.

2.4.1.6. Related Representations

Within the context of signal theory, independent research has produced other multiresolution signal representations based on quadrature mirror filters. Adelson and Simoncelli[54] have studied a number of orthogonal pyramid transforms based on variously shaped generating kernels. Besides separable QMF with supports in the range of five to nine tapes, they also consider nonseparable decompositions, such as those derived from hexagonal tessellated images, and the so-called quincunx pyramid, which instead partitions the frequency spectrum into nested diamonds and squares.

Relaxing the requirement of orthogonality while keeping the property of completeness in signal reconstruction makes other pyramid transforms possible. The Laplacian pyramid by Burt[39] is the most popular. It is covered in detail in Section 2.4.3.

2.4.2. Gaussian Pyramid

A common multiscale representation is the Gaussian pyramid. The advantages of this representation range from the mathematical properties of the Gaussian function to correspondence with an early vision operator in humans. As stated by many authors the appealing mathematical properties are that the Gaussian function can be iteratively built on the basis of the simplest functions exploiting the central limit theorem (Wells[56] explicitly addresses this property to obtain the Gaussian function by using a cascade of uniform filters). This being a two-dimensional rotational symmetric function, it not only maintains its shape in the frequency domain, but it is also decomposable in the product of two monodimensional Gaussian functions; it is the only filtering function that, by operating in coarse-to-fine mode, preserves the extrema: maxima and minima are never removed; eventually new ones can spring out.[57] In particular, as proved by Yuille and Poggio, Gaussian filtering "does not create generic zero crossing as the scale increases."[58] The importance of Gaussian-like operators in the early vision phase was first highlighted by Marr and Hildreth,[59] who proposed the adoption of a sequence of Gaussian filtering with module $M = 1.6$ in order to prepare the "primal sketch."

The Gaussian pyramid represents images in multiresolution by a sequence

of low-pass filtered images $\{G_0, G_1, G_2, G_3, \ldots, G_n\}$ obtained by convoluting the original image with a set of Gaussian kernels that differ in size by multiples of 2 and regularly subsampling by discarding a number of rows and columns equal to $2^i - 1$. According to the multiresolution theory of Mallat, this pyramid is a multiresolution approximation built on a nonorthonormal basis: many of the analytical derivations which follow can be easily brought back to the wavelet framework. We shall refer to the series $\{G_0, G_1, G_2, G_3, \ldots, G_n\}$ as the *Gaussian pyramid* **G**. The lowest level G_0 is the original image I. If I has equal linear size of $2^n + 1$ pixels, then the complete pyramid will have just $n+1$ levels (more generally, the same pyramid data structure of $n+1$ levels can be constructed from an image of $M\ 2^n + 1$ rows and $N\ 2^n + 1$ columns, M and N being integers).

For $h > 0$, G_h is given by

$$G_h(i,j) = \sum_{r,s=-k_h}^{r,s=+k_h} \sum W_h(r,s) G_0(i2^h - \mathrm{r},\ \mathrm{j}2^h - \mathrm{s}) \qquad \forall h,\quad 1 \le h \le n \tag{2.10}$$

where W_h is the *hierarchical kernel* corresponding to level h, and $2k_h + 1$ is its linear spatial dimension.

Still using the notation introduced by Burt,[60] the same expression in terms of the convolution operator $(*)$ can be stated more concisely as

$$G_h = [W_h * G_0]_{\downarrow 2^h} \qquad \forall h,\quad 1 \le h \le n \tag{2.11}$$

where $[\cdot]_{\downarrow 2^h}$ indicates a subsampling of a factor 2^h of the image contained between brackets.

The most convenient way of constructing **G** is to iterate the generation from each level h to the next $h+1$ by convoluting G_h with the same low-pass filter w, which is called a *generating kernel* and discarding the even rows and columns:

$$G_h = [w * G_{h-1}]_{\downarrow 2} \qquad \forall h,\quad 1 \le h \le n \tag{2.12}$$

The selection of w is a very important issue; three main aspects must be considered: its extension, its shape, and its computational cost.

- The width of the support used to build each new level (receptive field) is defined as a trade-off between precision in kernel shape definition and its own computational cost, obviously, also on the basis of original image quality

and of the tasks of analysis at hand. The hardware support built for this purpose up to today ranges from the 2×1 or 1×2 array kernel of the nonoverlapped bin pyramid (see Section 5.2.4.4) to the 5×5 kernel of some multiresolution systems (see Chapter 6). In what follows, we refer to this last array size without explicit annotation. Note that, the effect of maintaining constant w is to double the equivalent number of pixels of the original image for each linear dimension when generating each new level.

- The solution of iterating the construction of **G** from level to level using a fixed generating kernel is considerably more efficient than the original convolution with I, but this means that the actual equivalent shape of the kernel is changing from level to level and can only coarsely approximate the assumed Gaussian shape (this is particularly true for the lowest levels of the pyramid). A more precise analysis of this aspect is given in the next section; nevertheless, let us anticipate that the most crude approximations correspond to those levels for which human eyes are also mostly insensitive to contrast perturbations.[61]
- N and $2k+1$ being respectively the linear size of I and of the generating kernel, the cost of the Gaussian pyramid construction is on the order of N^2k^2. To limit this computation cost, following Burt,[39] w is chosen with four constraints. First of all it has to be *small* and *separable:*

$$w(r,s)=\hat{w}(r)\hat{w}(s) \tag{2.13}$$

Second, each monodimensional component is *normalized:*

$$\sum_{r=-k}^{k}\hat{w}(r)=1 \tag{2.14}$$

Third, each component is *symmetric:*

$$\hat{w}(-r)=\hat{w}(r), \qquad -k\le r\le k \tag{2.15}$$

Finally, each pixel of the previous level must give the same total weight for the construction of a new level *(principle of equal contribution).* Considering discard when subsampling the even rows and columns, this means that the total weight of the odd position in the kernel must equal that of the even position:

$$\hat{w}(0)+2\sum_{r}\hat{w}(2r)=2\sum_{s}\hat{w}(2s-1)=\frac{1}{2}, \qquad (2.16)$$
$$1<2r\le k, \qquad 1\le 2s-1\le k$$

where r and s are integers, and the first three constraints have been satisfied.

In particular, for $k=2$, $\hat{w}(1)=\hat{w}(-1)=0.25$, and $\hat{w}(2)=\hat{w}(-2)=0.25-\hat{w}(0)/2$: just one independent variable remains; its influence on the kernel shape and on the general properties of the resulting pyramid will be analyzed for this special common case at the end of the next section. From the computation cost point of view, the construction of the complete **G** pyramid requires approximately 10 operations per pixel of the original image.

2.4.2.1. Equivalent Weighting Function

In order to study the analytical properties of the ensuing pyramid, let us investigate the relationship between the hierarchical kernel W_h and the generating kernel w. The explicit relationship between successive levels G_h and G_{h-1} [given in compact form in Eq. (2.12)], under the hypothesis that all subsampling operations will be performed at the end of the convolution steps, and with the coordinate (i, j) referred to G_0, is

$$G_h(i,j)=\sum_{r,s=-k_h}^{r,s=+k_h}\sum w(r,s)G_{h-1}(i-r2^{h-1}, j-s2^{h-1}) \qquad (2.17)$$
$$\forall h, \quad 1\le h\le n$$

that is, w component spacing is doubled from level to level as is its linear size.

The *expanding kernel* w_h applied to level $h-1$ to construct level h can be expressed in analytical form as follows:

$$w_h(r,s)=\begin{cases} w\left(\dfrac{r}{2^{h-1}},\dfrac{s}{2^{h-1}}\right) & \text{for } r \text{ and } s \text{ multiples of } 2^{h-1} \\ 0 & \text{otherwise} \end{cases} \qquad \forall h,\ 1\le h\le n \quad (2.18)$$

The *equivalent weighting function*[60] which results from the cascade of these expanding kernels is then expressed by the following hierarchical kernel W_h:

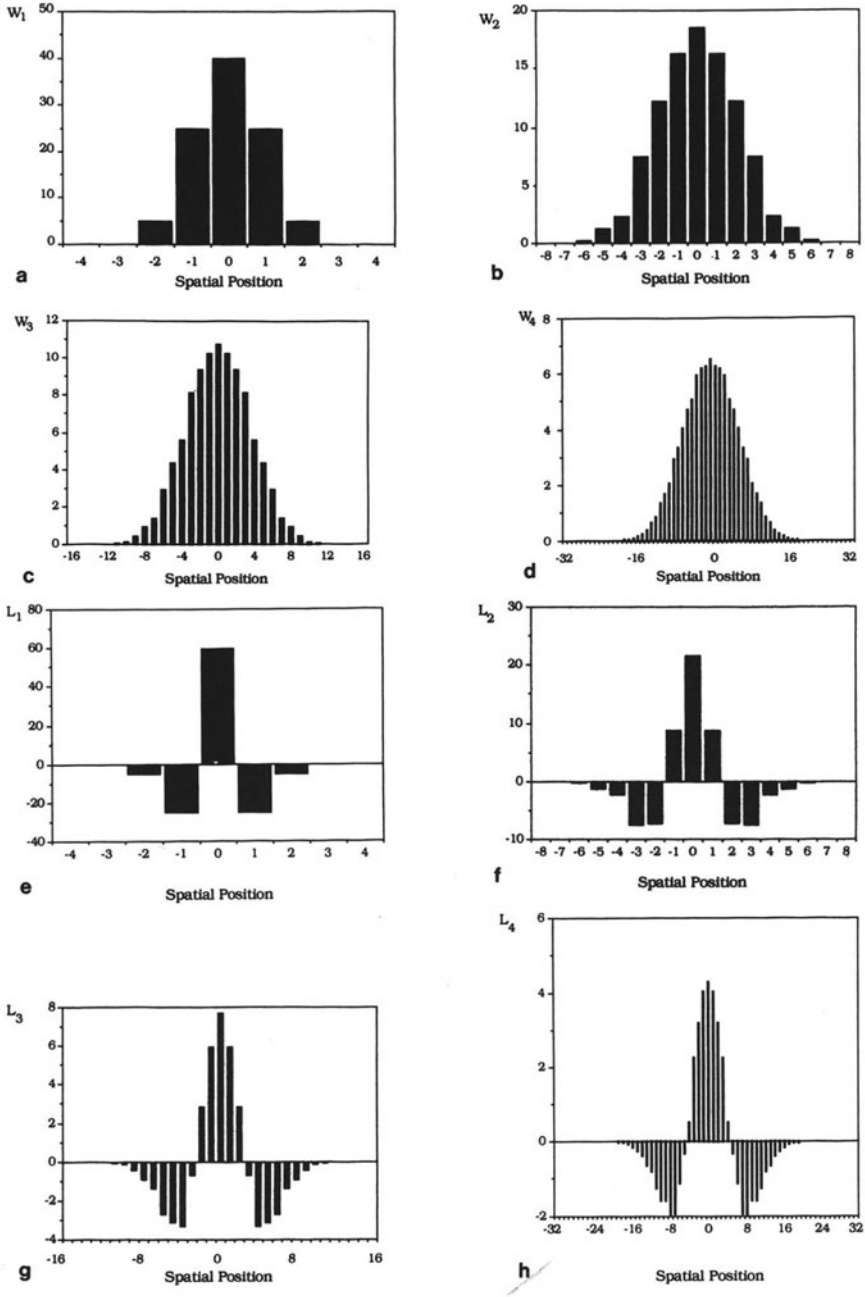


Figure 2.7. The first four levels of the equivalent weighting functions of G (a–d) and L (e–h) constructed iteratively with $\hat{\mathbf{W}}_1(0) = 0.4$ and under the constraints represented by Eqs. (2.13–2.16). Note that in the fourth level the curves closely resemble the Gaussian density function and the common Laplacian operator used in edge detection.

$$W_h = w_1 * \ldots * w_{h-1} * w_h \tag{2.19}$$

[remember that W_h is the kernel that, if convoluted directly with the original image I, gives the same values at level h achieved by the h iterations (2.18)]. The resulting characteristics of **G** can be analyzed on the basis of this equation, which summarizes the effect of the iterative construction. In Figure 2.7a–d the spatially normalized plots of the hierarchical kernels $\hat{W}_1$, W_2, $\hat{W}_3$, $\hat{W}_4$ are given for $k=2$ and $\hat{w}(0)=0.4$ [consequently, $\hat{w}(1)=\hat{w}(-1)=0.25$ and $\hat{w}(2)=\hat{w}(-2)=0.05$]. As can be seen, the higher the level the closer the resulting shapes to a proper Gaussian function (it is claimed that for $h\rightarrow\infty$ the hierarchical kernel becomes a Gaussian function).

The importance of good selection of the generating kernel parameters is indicated in Ref. 39, where the case for $k=2$ is analyzed in detail. As cited earlier, in this case the constraints expressed in Eqs. (2.13)–(2.16) leave just one independent parameter that can be selected by assigning a value to $\hat{w}(0)$. The paper quoted introduces the concept of how the overall characteristics of **G** closely depend on this assignment. In particular, it is shown how the values 0.5–0.4 correspond respectively to an equivalent weighting function with triangular ($\hat{W}_1=\{0.5\text{–}0.25\text{–}0.0\}$) and broadly Gaussian shapes on all levels of the pyramid. Conversely, for compact coding purposes it can be shown that the best value of the generating parameter is $\hat{w}(0)=0.6$, and it can be shown as well that with a suitable quantization process, a simple (from the computational point of view) and effective coding scheme can be derived[39] from the consequent values in **G**.

2.4.2.2. The Expand Operator

An operator which reverses the generating kernel w must be introduced in order to congruently compare data resulting at two different levels at the highest detail. This operation (which has been called a *projection* by Hanson and Riseman,[62] while here we follow the Burt denomination) can be defined, as in (2.11), in terms of the convolution operator (*) as follows:

$$G_{h,t} = [W'_{h,t} * G_h]_{\uparrow 2^t} \qquad \forall h,\ 0 \le h \tag{2.20}$$

where $W'_{h,t}$ is called an *expanding kernel*, and $[\cdot]_{\uparrow 2^t}$ explicitly indicates the interpolation by a factor 2^t of the initial image G_h, so the resulting sizes of $G_{h,t}$ will be $\{2^tN+1,\ 2^tM+1\}$ ($\{N+1,\ M+1\}$ are the sizes of G_h). Note that

$G_{h,0}=G_h$, and $G_{0,0}$, $G_{1,1}$, $G_{2,2}$, . . . , $G_{n,n}$ will have the same sizes as the original image I.

Just as for the hierarchical kernel, the most convenient way of constructing $G_{h,t}$ is to iterate the interpolation operation step by step, doubling in the linear spacing dimension:

$$G_{h,r}=[w' * G_{h,r-1}]_{\uparrow 2} \qquad \forall r, \quad 1\le r\le t \tag{2.21}$$

This can be implemented[12] by padding with "zeros" (inserting new even rows and columns with null samples) and convoluting the resulting image with a suitable filter w'. Explicitly, the first step is

$$\tilde{G}_{h,r}(i,j)=\begin{cases} G_{h,r-1}(\lfloor i/2\rfloor,\lfloor j/2\rfloor) & \text{for } i \text{ and } j \text{ even} \\ 0 & \text{otherwise} \end{cases} \qquad \forall r,\ 1\le r\le t \tag{2.22}$$

The expanding kernel w' is usually selected on the basis of the same constraints of the generating kernel as expressed by Eqs. (2.13)–(2.16). Thus, the second step will be simply

$$G_{h,r}=4\{w' * \tilde{G}_{h,r}\} \qquad \forall r, \quad 1\le r\le t \tag{2.23}$$

where the factor 4 is introduced to compensate for the fact that three out of four samples in $\tilde{G}_{h,r}$ are zeroes.

2.4.2.3. Boundary Conditions

For generating and expanding data some care must be taken regarding the border of the image, especially when the kernels are large. The usual approach is to extend the current image G_h in the border, preserving the continuity of the image itself and of the first $k-1$ derivatives. As a common example, for $k=2$, G_h is augmented by two rows or columns for each edge border, in which values are assigned by reflection and inversion across the edge.[61] Since $G_h(i,0)$ $(0\le i\le N)$ are the first columns, the values assigned at the two left extra columns will be

$$\left| \begin{array}{l} G_h(i,-1)=2G_h(i,0)-G_h(i,1) \\ G_h(i,-2)=2G_h(i,0)-G_h(i,2) \end{array} \right| \qquad \forall i, \quad 0\le i\le N \tag{2.24}$$

In this way the first derivative is constant at the left boundary (second derivative is zero). The behavior on the other sides is derived from this.

2.4.3. Laplacian Pyramid

In **G** the image is reproduced many times by reducing the frequency content one octave at a time with low-pass filtering step by step. An alternative image representation, popular in the image processing community, is given by the Fourier transform, which uses the spatial frequency distribution to characterize the image content. Both these image data representations undersupply some frequent analyses: the Fourier transform is inappropriate when the *spatial location* of patterns is critical; the Gaussian pyramid is unsuitable when tasks require *spatial-spectrum*-based analysis. A pyramidal representation which retains both spatial localization and spatial frequency composition information is given by a generating kernel which implements a local bandpass filtering operation.

As for **G** construction, operators of identical shapes, but with different scales, are applied step by step. As in the quoted case, these operators are local so as to preserve spatial information and are localized in the spatial frequency domain by reducing one octave both the frequency center of the bandpass and the bandwidth (i.e., this is achieved by doubling the generating kernel linear size in the spatial domain).

From the cost computation viewpoint the cheapest solution to achieve such a representation is to manipulate **G** (which is already selective in spatial frequency content). This solution has a general framework in the difference of low-pass transform (DOLP).[63] Since the first octave of the spectrum is directly given by G_n, the levels of the bandpass pyramid $\mathbf{L}=\{L_0, L_1, L_2, \ldots, L_n\}$, may be defined in terms of the low-pass pyramid in this way:

$$\begin{aligned} L_n &= G_n \\ L_h &= G_h - G_{h+1,\,1} \qquad \forall h, \quad 0 \le h < n \end{aligned} \tag{2.25}$$

The pyramid **L** corresponds closely to the wavelet decomposition of Eq. (2.8). However, note that this representation is redundant by one third, because its generating functions are not orthogonal.

2.4.3.1. Equivalent Weighting Function

Even in this case, it is convenient to define the equivalent weighting function as a kernel that if convoluted directly with the original image I gives the

same results as Eqs. (2.25). It is easily derived that this function is given by the difference of the Gaussian-like functions of two consecutive levels, precisely:

$$L_h = \left(W_h - w' * W_{h+1}\right) * I \qquad \forall h, \quad 0 \leq h < n \tag{2.26}$$

in which the hierarchical kernels are given by Eq. (2.19) (under the hypothesis that the subsampling operations are performed at the end of the sequence of convolution steps required to achieve each G component). In turn, they represent two equivalent weighting functions of the generation of $\mathbf{G}$. The kernel w' is one of the two suggested by Burt *et al.*[12] and has been called RE (reduce and expand) because the second term of the kernel in (2.26) is expanded after decimation. An alternative is presented in detail in Chapter 6, which does not require the expansion operation: the subtraction is performed when building the next Gaussian level before decimation; this second solution is called FSD (filter, subtract, and decimate). Nevertheless, with the FSD solution the property of completeness in image recovering (that will be subsequently described) is lost. Hence, in what follows we will always refer implicitly to the RE solution. Because of this interpretation of a set of differences between pairs of successive multiresolution Gaussian representations, with each one being more detailed than the next, the sequence obtained by (2.26) has been referred to as a pyramid of error images.[39]

These resulting functions closely resemble the well-known Laplacian operator ∇^2, introduced by Marr and Hildreth[59] for edge detection purposes, that operates in humanlike mode, following a multiresolution lateral inhibition approach. This is the reason why the sequence $\mathbf{L}$ is referred to as a *Laplacian pyramid*. In Figures 2.7e–h the spatially normalized plots of the first four hierarchical kernels are given for $k = 2$ and $\hat{w}(0) = 0.4$. In order to facilitate the computations these solutions are pursued following the FSD approach. The behavior is similar to that of the $\mathbf{G}$ case: note that the higher the level the closer the resulting shapes are to those of the Laplacian operator.

2.4.3.2. Original Image Recovering

From Eqs. (2.25) and (2.20) it can be easily derived that the original image I can be exactly recovered from the Laplacian pyramid by expanding and summing all levels:

$$I = G_0 = \sum_{h=0}^{n} L_{h,h} \tag{2.27}$$

i.e., the Laplacian pyramid is a complete image representation (like the wavelet representation), and the local operators of many scales but identical shapes, introduced thus far, serve as the basic functions in this representation.[64] Furthermore, this representation enhances image features like edges, which play a very important role in image analysis.

Another procedure to recover the original image more efficiently from the computational point of view is given by reversing the order sequence of the Laplacian pyramid generation. The topmost level L_n is expanded and added to L_{n-1}, then this last one is expanded and added to L_{n-2}, and so on, until level 0 is reached. By adding L_0, is added to the original image until G_0 is achieved.

2.4.3.3. Remarks on L

The use of multiresolution for detecting edges is a well-established approach and can be traced to the work of Roseneld and Thurston.[65] The persistency of real edges (as opposed to fake ones) through the resolutions as highlighted by the Laplacian operator has been suggested as the key issue for scene segmentation by Witkin[66] in his ''scale-space'' approach.

The topmost-to-base process of recovering G_0 can be applied to achieve a progressive image transmission[39] in which initially a coarse sketch of the image content is given, following which, if required, a subsequent transmission provides the details at the requested resolution. Moreover, it has been proven that humans are more sensitive to error in the low- (and in medium) spatial-frequency components and they are relatively insensitive to the high spatial frequencies. But three fourths of the data of **L** belongs to L_0, and fifteen sixteenths to the first two levels. On this basis, a quantization scheme which takes care of this different behavior was introduced by Burt in the cited paper. In practice, for L_0 and the first levels, where the sample density is fine, coarse quantization levels are applied (and consequently a smaller bit-to-pixel ratio); conversely, at the coarse sample density of the higher levels, finer quantization levels are needed. In Ref. 39 a few common examples are given, in which the image quality is guaranteed with data rates around 1 bit/pixel (and the corresponding mean square error of less than 1%). For image motion analysis purposes it has been found[13] that a ratio of 3 bits/pixel does not noticeably degrade the computer flow field.

Some conjectures that the human visual system uses these kinds of repre-

sentation have been reported in the scientific literature.[60] This representation has been effectively applied in applications, like image mosaicking,[61] in which the sensitivity of the human vision constitutes the figure of merit for the evaluation of the results.

2.4.4. Haar Pyramid

Though defined independently of the multiresolution approach, the simplest orthogonal pyramid transform is the Haar transform.[67] Originally, it was defined in the monodimensional case; it has been extended with some modifications to the bidimensional domain.[68]

In this last version the transform is a wavelet decomposition, based on the three generating wavelets Y_1, Y_2, Y_3, shown in Figure 2.8a. This is an instance of a very simple wavelet transform, since the wavelet coefficients use only the three values $(1, 0, -1)$. Figure 2.8b shows the first 64 basic functions as produced by the three generators at the first three resolution levels. The generators are sensible to step edges in images, and this has stimulated a certain interest in such a transform for image processing and coding since its first introduction.

The definition of the Haar transform in the bidimensional discrete space is straightforward. A square image of $2^n \times 2^n$ pixels has a discrete Haar transform at $n+1$ resolutions (including the dc component). At any scale i $(0 \leq i \leq n-1$, excluding the dc component), there are 2^{2i} functions having a support of $2^{2(n-i)}$ pixels and taking on, within their support, one of the two values $\pm 1/2^{(n-i)}$. A proper rescaling of coefficients according to the resolution i allows us to code the transform with elementary arithmetic based on the three coefficients 1, 0, -1.

There are three motivations for introducing the Haar transform for vision problems: the first is that it is a wavelet transform with spatial and frequency localization; second, the support of the generating kernel perfectly matches the quaternary-tree data structure; third, the computational cost is reduced to less than three additions or subtractions per pixel of the input image.

2.4.4.1. A Pyramid Implementation

The recursive use of the generators at different resolutions allows us to structure the computation of the coefficients hierarchically. Two approaches are possible: the former builds a multiresolution representation by averaging with a 2×2 kernel and then computes the transform conceptually in a single step at

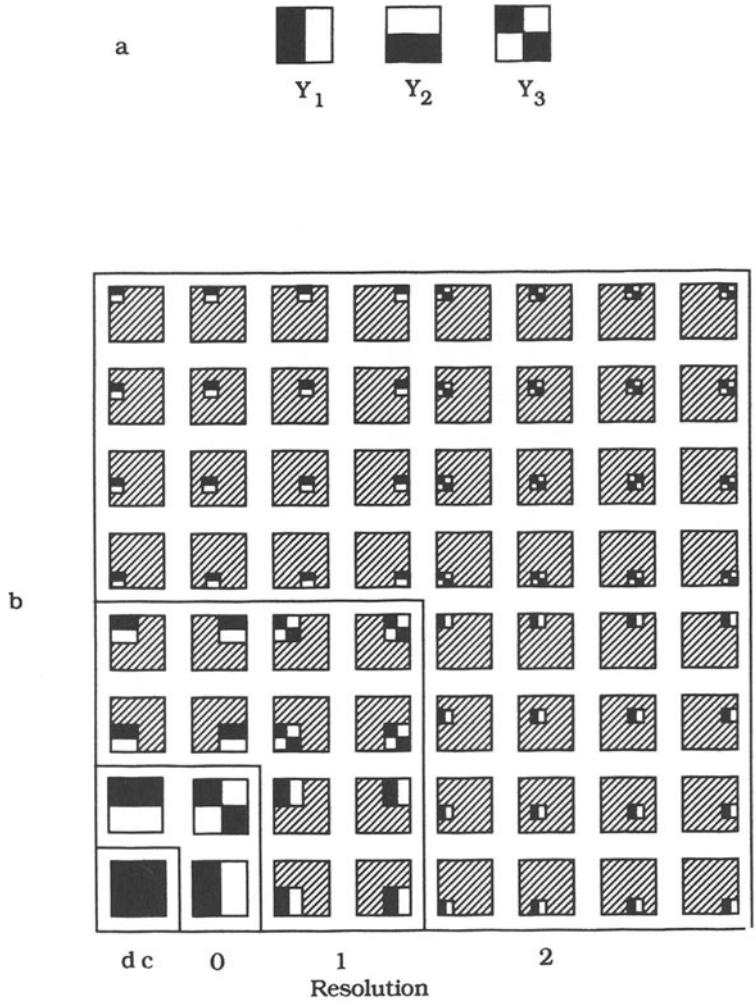


Figure 2.8. The bidimensional Haar transform as a wavelet: (a) the three generating wavelets; (b) the first 64 basis functions; according to the resolution they are grouped into three families of 4, 12, 48 elements; the coding of coefficients is black = 1, dashed = 0, white = −1.

all resolutions; the latter combines the building of each new reduced resolution image with the computation of the transform.

A previous work[69] has shown how to implement the Haar transform using a quad-pyramid data structure. For an $N \times N$ input image, let us consider a quaternary-tree data structure of $n+1$ levels ($n = \log_2 N$), where each leaf (node in the base) corresponds to one pixel of the input image. Note that at each level each node interacts during the computation only with its siblings, while hierarchical connections are used for interlevel data transmission. The algorithm for the direct Haar transform (DHA) (slightly modified here in the use of the coefficients) consists of two main phases.

1. *Construction of a set of images at different resolutions.* Except for the base, which stores the full original image, each node in the other levels of the pyramid builds a value which is the weighted sum of the pixels stored in its four children. By choosing all weights equal to 0.5, the formulation of the direct and inverse transforms as a local computation on all levels (second step) is identical. The setting up of the averaged images takes n steps.

2. *Local computation of the Haar operator.* In each level, 2×2 blocks of nodes are the local support of the operator. If A, B, C, and D are the values of the pixels in each block (respectively of the node located in the block at position northwest, northeast, southwest, and southeast), three output values, β, γ, δ, are computed according to the generators Y_1, Y_2, and Y_3. They are stored in the northeast, southwest, and southeast node of each block:

A	B
C	D

	β
γ	δ

$$\begin{aligned} \beta &= \tfrac{1}{2}(A - B + C - D) \\ \gamma &= \tfrac{1}{2}(A + B - C - D) \\ \delta &= \tfrac{1}{2}(A - B - C + D) \end{aligned} \tag{2.28}$$

The local computation is performed on each level of the pyramid (including the base), and the Haar transform consists of the collection of all sets of (β, γ, δ) values, plus the average value α stored in the apex of the pyramid (see Figure 2.9). We note that, while the multiresolution averaged version of the image is redundant by roughly a factor of $\frac{1}{3}$ (the quaternary-tree data structure is made up with $(4N^2 - 1)/3$ nodes), the transform is defined by exactly N^2 values, though distributed on the different levels: $\frac{3}{4}N^2$ in the base, $\frac{3}{4}(\frac{1}{4}N^2)$ on the next level, etc.). Moreover, note that the values computed according to this algorithm do not require rescaling but already comply with the orthonormal condition in the discrete case.

The inverse Haar transform (IHA) is a top-down algorithm. It assumes that

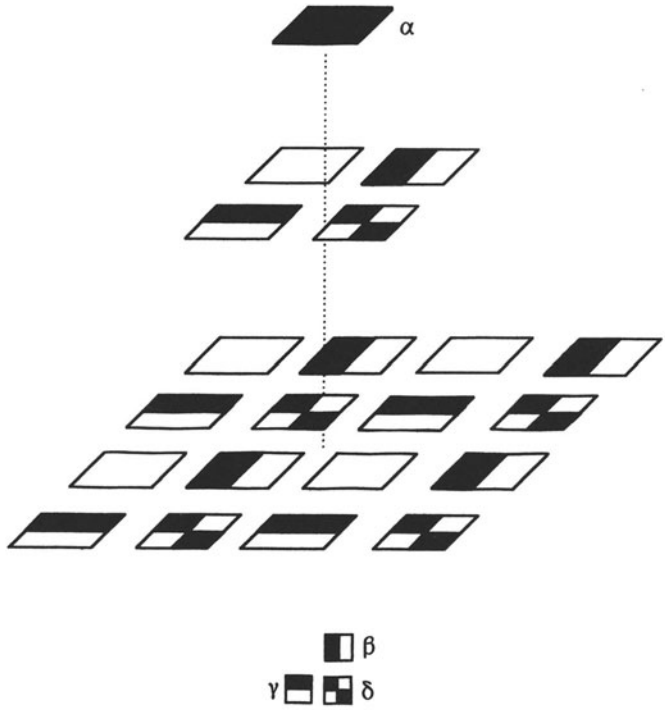


Figure 2.9. The layout of the quaternary-tree-based Haar transform of a 4×4 image. Empty boxes denote unused nodes.

the values to be antitransformed are stored in the levels of pyramid according to the outcome of the DHA. Starting from the apex, the value of the parent (the dc component) is copied in the northwest node within the 2×2 block of its children; let us denote such a value by α. The $(\alpha, \beta, \gamma, \delta)$ values in the block are then transformed according to the following equations:

$$\begin{aligned} A &= \tfrac{1}{2}(\alpha+\beta+\gamma+\delta) & C &= \tfrac{1}{2}(\alpha+\beta-\gamma-\delta) \\ B &= \tfrac{1}{2}(\alpha-\beta+\gamma-\delta) & D &= \tfrac{1}{2}(\alpha-\beta-\gamma+\delta) \end{aligned} \tag{2.29}$$

The same three generators Y_1, Y_2, and Y_3 used in the direct transform are applied to the reconstruction process, plus a Y_0 generator to produce the average coefficient A. At the next stage of the top-down process, the A, B, C, D values are propagated downward to become the α values of the blocks in the

level below. When the propagation of the data and the computation reach the basis, the IHA is complete and the original image is reconstructed in the base.

An alternative solution follows the approach of using the same generating kernels step by step. The DHA computation can be recast into a pipeline of processes, each active on successive levels of the pyramid starting from the base. These processes combine the construction of the averaged version of the image and the local computation into a local transformation followed by a data reduction. With the same conventions for naming the values within each 2×2 block, the local transformation carried out consists of the three generators Y_1, Y_2, and Y_3 of Eq. (2.28) plus the averaging generator Y_0, yielding α:

$$\alpha = \tfrac{1}{2}(A + B + C + D) \tag{2.30}$$

During the reduction step, the α value of each 2×2 block is sent to the corresponding parent node, thus generating the upper-level image. The process of local transformation and reduction is iterated till the apex is reached.

2.4.4.2. Other QMF Transforms

As previously mentioned, the Haar transform is an instance of a wider class of orthogonal transformations, the *quadrature mirror filters* (QMF). For such filters, both odd-type and even-type kernels have been investigated. In order to obtain good performance in frequency discrimination, one-dimensional kernels of nine taps and larger must be used.[54] Concerning this point, the two-tap shape of the Haar basic functions is considered poor.

The interest here for quadrature mirror filters is due to their intrinsically hierarchical nature, as mentioned in Section 2.4.1. They give rise to pyramid orthogonal algorithms relying on a wide support combined with a standard quaternary pyramid with a reduction factor of 4. They use therefore a variation of the pyramid structure known as an *overlapped* pyramid; the value of a parent node is obtained by a weighted average of all its children, and the sets of children of two neighboring nodes are not disjoint. This condition ensues when the support used to generate a node at the next resolution level is larger than a 2×2 block. Quadrature mirror filters with a 3×3, 5×5, . . . bidimensional kernels are instances of the case (obviously, even-sized kernels also have the same effect, starting with 4×4).

The quad-pyramid computer (see Chapter 5) has been shown[70] to be able to build overlapped pyramid representations of any shape without appreciable overhead, both in the bottom-up and top-down processes.

The data-parallel implementation of the Haar transform is therefore the

simplest case of a much wider class of QMF transforms for which the pyramid is the best theoretical data structure. Besides it can be used very profitably for foveation, tracking, and general focusing strategies as well as for image coding.

2.4.5. Feature Pyramid

In an ordinary pyramid, images are represented by a sequence of copies of the original data in which resolution is decreased by regular steps, from the base (containing the original image) toward the apex. Many authors (for a survey see Dyer[32]), extended the pyramid concept to other points or local properties, like color, edge density, energy measure of any feature detector (see the next section for an example), etc.

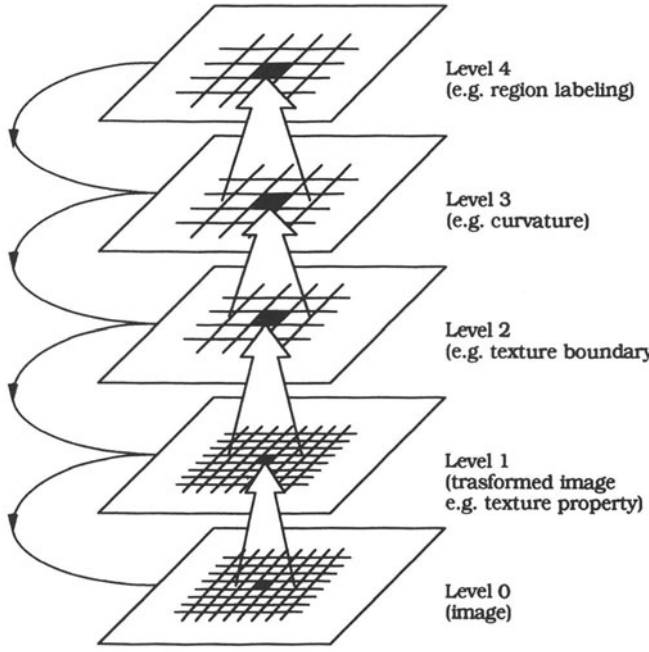


Figure 2.10. A four-level example of a feature pyramid. The base is the original image. The successive layers show some steps of a segmentation process based on a texture analysis. Namely, the first level is the image resulting from a transformation produced by a texture detector; the second and the third, respectively, represent the border and the curvature of the boundary of the segments; the fourth is the regions identified and labeled.

Color pyramids, that in general construct a pyramid (Gaussian, Laplacian, etc.) for each primary color, were introduced in 1980.[71] A second very general example of feature pyramid is built on the energy measure of a feature detector. Having in some levels k a local description of the density of a property given by a feature f, the Gaussian pyramid built over this level G_k will give the Gaussian integration of the feature f level by level on a larger and larger subarray. The final decision on the existence of a scale-dependent property can be based on the value of the integration G_k.

Granlund and Arvidsson in 1983 (Ref. 72) introduced the term *feature pyramid* for the hierarchical heterogeneous data structure shown in Figure 2.10. The base of the pyramid is the original image. The local feature detector produces the first level (it does not matter what kernel size and decimation are applied). A second, generally different, operator is applied to this level to produce the successive level. From level to level different transformations in a bottom-up sequence produce new local evidence of the properties of the original images. The bottom-up construction represents the partial result at the different levels of the problem. The levels form a ''semantic'' pyramid, and each one contains information about properties or features that are derived from the preceding levels. In fact, the top-down activity shown on the left of the figure represents a feedback control of the higher levels over the operations in execution on the lower one. A system (GOP-300) which is specialized in this approach for image analysis was designed and commercialized by the quoted authors. A few details on this machine will be given in Section 6.4.

2.5. COARSE-TO-FINE SEARCHES

As mentioned, when a property is spatially invariant across the scale, it can be effectively (from a computational point of view) applied to a coarse-to-fine approach. It consists of two basic steps: (i) the selection of a low-level scale to hypothesize the presence of what we are looking for; (ii) the zooming in on the exact position at a higher scale to achieve evidence of the presence or absence of the target.

The determination of the optimum level at which the search has to start depends on the characteristics of the task at hand. Note that this is a critical selection; instances missed at this level will never be recovered even if there is substantial evidence at the finer levels. In general, the selection obviously depends on the original spatial resolution; e.g., the maximum shift in position allowed to the target in the search space, etc. For reasonable values of these

factors, the number of levels used turns out to be less than or equal to 3 or 4, which corresponds to a data reduction factor of at least 64 or 256 (to which obviously a higher speedup may correspond if the task at hand is more than linear on the amount of data).

Once the result of the first screening of the region of interest is obtained, the second phase of refinement has to be performed; here the best strategy is to proceed top-down through iterative rearrangements; that is, the refinement must be accomplished step by step at every intermediate level until the maximum resolution level is reached. At each level the confidence of the selection is checked, and the spatial position of scrutiny is refined by checking (e.g., by thresholding a suitable figure of merit) the g near neighbors corresponding to the position of the previous level. The computation overhead required by the refinement phase is dependent on the number of resolution levels, s, involved.

Considering the matching example introduced in Section 2.3, let us suppose that we are looking for targets sized T^2 in the base of the pyramid (the cost of computation of a direct correlation will be N^2T^2). Since s is the number of successive scaled images, the cost of computation of the first step will be

$$\dot{O}_{\text{ctf}} = O\left(\frac{N^2T^2}{M^{2(s-1)}}\right) \tag{2.31}$$

Since p is the number of regions of interest detected, the additional factor for the focusing processes is

$$\begin{aligned}\ddot{O}_{\text{ctf}} &= O\left(pg\left\{\frac{T^2}{M^{2(s-2)}} + \cdots + \frac{T^2}{M^2} + T^2\right\}\right) \\ &= O\left(pgT^2\frac{1-M^{-2(s-1)}}{1-M^{-2}}\right)\end{aligned} \tag{2.32}$$

Even if this second component increases with the number of reductions in resolution, the overall cost monotonically decreases for reasonable values of s: for $M=2$, $p=8$–16, and $g=10$ the gain is by a factor of 4 for $s=2$ and around 15 and 200 for s equal to 3 and 4, respectively.

Another feature of the coarse-to-fine approach is flexibility; if the evidence achieved is sufficient, the refinement phase can be stopped before reaching the maximum resolution level. Once again the computational speedup can become remarkable, even if a certain penalty is paid in precision.

The first, and best-known, example of a coarse-to-fine process is the edge

detection algorithm introduced by Kelly[73] in 1971. This algorithm can be considered a general paradigm or the focusing procedure and is described in detail in the sequel. It is composed of the two functions **edges** and **refine.** In the first at an initial level **I,** all the pixels are checked to verify if the feature detector (in this case the function **boundary**) is greater than a given threshold. Only in these cases is the second function **refine** called. The number of edge points is obviously a small fraction of the total number of pixels in level **I**. Only in the detected positions does the procedure **refine** look for the precise location of the edge recursively, till the finest resolution is reached, inspecting the subarray of size **template.**

```
void edges (pyramid *inp, pyramid *outp, int I)
{  int  i,j;
 double edge;
 for (i=0; i<pow (2,I); i++)
        {  for (j=0; j<pow (2,I); j++)
              {  edge=boundary (inp, I, i, j);
                 store (oupt, I, i, j, edge);
                 if (edge>threshold)
                   refine (inp, outp, I, i, j);
              }
        }
}

void refine (pyramid *inp, pyramid *outp, int I, int i, int j)
{  int di,dj;
   double edge;
   i*=2;
   j*=2;
   /*I=0 in the apex */
   if (I++<maxI)
        {  for (di=0; di<=template; di++)
              {  for (dj=0; dj<=template; dj++)
                   {  edge=boundary (inp, I, i+di, j +dj);
                      store (outp, I, i+di, j+dj, edge);
                      if (edge >threshold)
                        refine (inp, outp, I, i, j);
                   }
              }
        }
}
```

2.6. IMAGE FLOW DIAGRAMS

The image flow diagrams introduced here are graphical sketches to illustrate pyramid image processing algorithms at a glance. The convention that is here presented in an extension of one used by Burt *et al.* in several papers cited in this chapter and more extensively introduced in Ref. 74. The paths of the image data connect six basic component symbols: (i) buffers to store input, partial results, and output data; in particular, a full data set buffer is indicated by □, and a region of interest inside it (windowing operation) by ❑; (ii) processing elements for filtering and point-dependent operations are represented by a ○ containing an indication of the operation inside; furthermore, this symbol represents both subsampling and expand operations; in these case ↓ and ↑ appear, respectively, inside the circle in addition to a figure representing the linear ratio between the data amount in output and input; (iii) dyadic arithmetic and Boolean and comparison operations between images are represented by the symbol ▶ having inside the detail of the operation performed; (iv) pyramid generations are represented by Δ, and the type of generating kernels used are specified (Gaussian, Laplacian, Haar, etc.); (v) delay operations are indicated by ◘; the number of clock cycles of the delay are given inside the symbol; (vi) the refinement operation, represented by ➢; the refinement is performed up to the level reported outside the sign on the basis of a threshold.

2.6.1. Examples

To illustrate the use of image flow diagrams, we give some basic examples of image processing tasks (Examples 2 and 3 are derived from Ref. 3).

2.6.1.1. Example 1

The first example is the well-known edge detection technique codified in the C language in the previous section. The coding by an image flow diagram is given in Figure 2.11a. The figure is self-explanatory: from the original image a Gaussian pyramid is built, a function f_k implementing the edge detector is followed by the couple thresholding refinement to produce the contour image at the maximum resolution.

2.6.1.2. Example 2

The second example is related to a detection or classification task based on a single feature or pattern characteristic (template matcher, texture descrip-

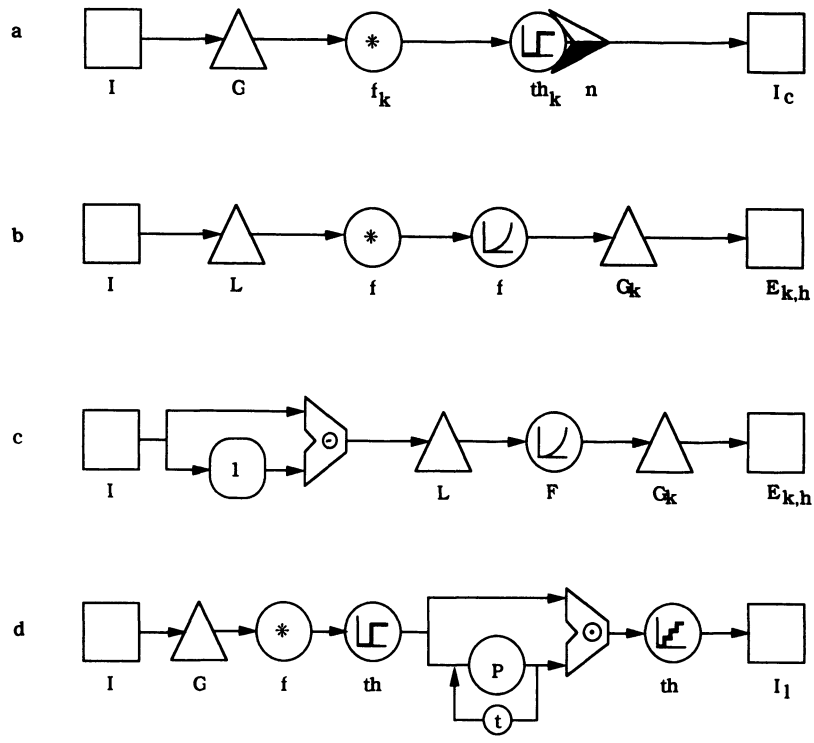


Figure 2.11. Four basic paradigms for algorithms of image analysis expressed in image flow diagrams: (a) a coarse-to-fine edge detector; (b) generation of integrated image feature sets; (c) simplified change-energy measures in a video sequence; (d) multiresolution labeling technique.

tor, spot detector, or others). Figure 2.11b shows the image flow diagram of the four stages for the generation of the image measures of the feature at hand for just one selected spatial band. In a first phase the Laplacian pyramid is built, and a filter which is selective of the feature chosen is applied. At this stage of processing, for each level of the pyramid, a point base measure is obtained. An enhancement step follows, e.g., the squaring of the measured parameter to prepare data for the final decision. Then, for the band(s) k that best highlights the feature of interest, a Gaussian pyramid is built to form different local energy measures by integrating locally, with various window sizes. The final result $\{E_{k,h}\}$, represents a set of measures of the energy of the chosen parameter for the given feature scale k and for different locations and window extensions h.

2.6.1.3. Example 3

The third example regards a simplified algorithm for motion analysis performed over couples of successive frames of a video sequence.[75] Figure 2.11c shows the image flow diagram of the five stages of an alerting procedure which measures the ''change energy'' for just one selected temporal and spatial band. In the first phase a difference image is formed by subtracting the current one with a delayed one, to detect points in which something is changing in the scene. Then the Laplacian pyramid is built, and a particular bandpass level is selected on the basis of the target's size and velocity (for example, high-frequency motion of leaves and grass or low-frequency motion of clouds or objects far away must not be detected; instead moving cars and humans must be considered). An enhancement step follows, e.g., a square of the measured parameter, to prepare data for the successive steps. Now, for the selected band(s) k that characterizes the motion under investigation, a pyramid is built, as in the previous algorithm, to integrate locally to form local change-energy measures. The resulting energy, $\{E_{k,h}\}$, can be used to guide the search for the target.

2.6.1.4. Example 4

The fourth example is a labeling pyramid technique. The objective is to label the border points, in each level of the pyramid, on the basis of the local curvature. The method used is the one described in Ref. 76. It consists of simulating a diffusion process: starting with an equal distribution of data on the contour, a standard heat diffusion process is originated on the contour toward the object inside (heat diffusion is supposed to be adiabatic with the background). During the transitory phase, the values on the contour are strictly dependent on the local curvature: in a few steps of heat propagation the local ''temperature'' on the border supplies a robust labeling for several curvature classes (the minimum number of labels was the five used in Section 2.2.1.3).

The first four blocks are what is used usually to construct a binary contour pyramid. The lower branch of the two parallel paths shows the t-recursive step of propagation P applied to all pyramidal levels. Here P just represents the following near-neighbor operation, applied for each pixel which belongs to the objects (T is null and so remains in the background):

$$T_{t+1} = T_t + K\left(\sum_{j=1}^{8} T_t[NN_j] - DT_t\right) \tag{2.33}$$

where T_i is the "temperature" at time i, K is the propagation coefficient, NN_j represents the near neighbor in position j under the hypothesis of Figure 5.4, and D is the number of near neighbors belonging to the object.

Finally the dyadic AND operator only selects the border pixels on which a multilevel threshold is applied to obtain the final contour labeled pyramid.

2.7. GENERAL PLANNING STRATEGIES

As stated in the first chapter an architecture is something more than a collection of elementary components. The goal of the designer is to gain performance with minimum effort (cost) by properly integrating different units. In designing a computer vision system some suggestions spring from knowledge of the human visual system. The approach in this case follows a general planning strategy which permits limited resources to overcome very complex tasks. The solution is achieved by means of a preanalysis, the result of which permits us to reformulate the problem in completely different terms and computational characteristics.

Figure 2.12 shows the basic framework of this planning strategy. On the basis of the pyramid theory (wavelet, Gaussian, Laplacian, etc.) a multiresolution environment is implemented. A smart sensing of the information at hand (outer cycle) is obtained with some specialized hardware consisting of a parallel implementation of a simple operator on the total amount of data (total field of view) at a low resolution. The objective of this peripheral (preattentive like) operation is to select the areas to scrutinize.

At this point, working with the full capability of the system at the sophisticated level of detail and by exploiting the *a priori* knowledge of what is going on or by performing a fast detection of the salient features map, the scene reality is interpreted and followed according to its own evolution.

When the *a priori* knowledge is insufficient, the detection of the spatial feature map is very important and critical. In this case, specialized hardware plays an important role. In fact, multiresolution systems supplying features at different scales at low cost, maintaining information locality, supporting correlation along the scales, and working both in coarse–to–fine and in fine–to–coarse modalities, allow us to solve tasks which in many practical cases overwhelm even the most sophisticated serial processor systems.

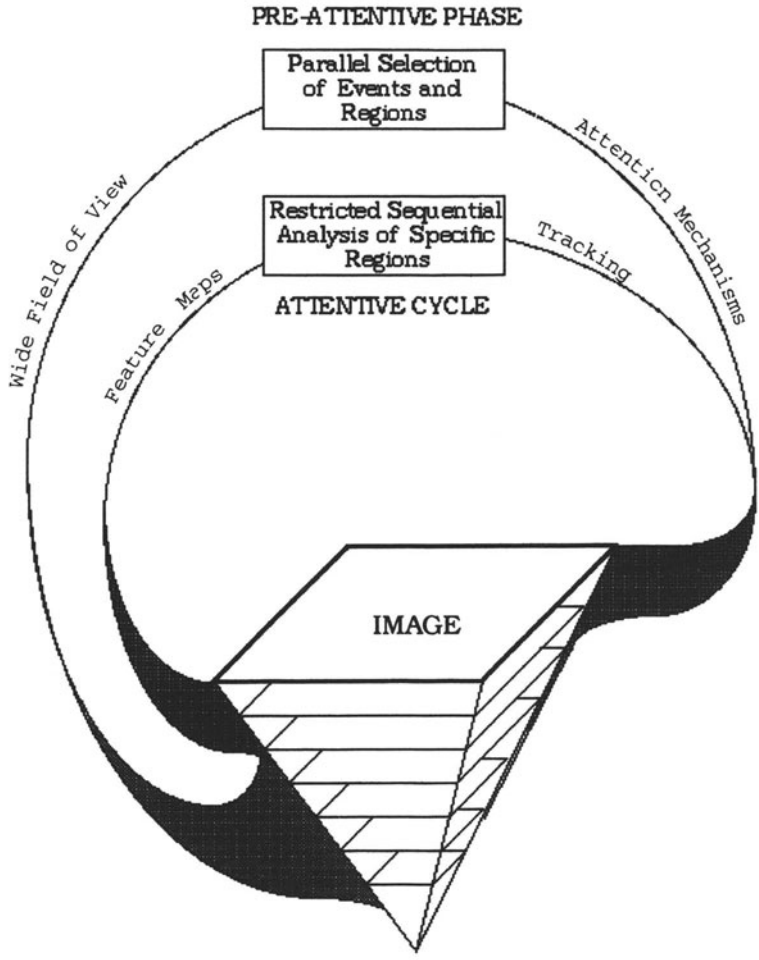


Figure 2.12. A planning strategy for a computer vision system. The outer cycle represents the attention mechanism followed by the proper scheduling of system resources. The inner level insists on the higher resolution areas but in limited regions.

REFERENCES

1 M A Fischler and O Firschein, *The Eyes, the Brain and the Computer*, Addison-Wesley, Reading, MA (1987)
2 L Uhr, Layered 'recognition cones' networks that preprocess, classify, and describe, *IEEE Trans Comput* **C-21** (7), 758–768 (1972)
3 P J Burt, 'Smart sensing' in machine vision, in *Machine Vision Algorithms, Architectures and Systems* (H Freeman, ed), pp 1–30, Academic Press, San Diego, CA (1988)
4 H Freeman, Machine vision approaches to automatic inspection, in *Progress in Image Analysis and Processing* II (V Cantoni, M Ferretti, S Levialdi, R Negrini, and R Stefanelli, eds), pp 601–615, World Scientific, Singapore (1992)
5 L Uhr, Highly parallel, hierarchical, recognition cone perceptual structures, in *Parallel Computer Vision* (L Uhr, ed), pp 249–287, Academic Press, London (1987)
6 J P Frisby, *Seeing Illusion, Brain and Mind,* Oxford University Press, Oxford (1980)
7 D H Hubel, *Eye, Brain and Vision,* Scientific American Books, New York (1988)
8 F A Geldard, *The Human Senses,* Wiley, New York (1972)
9 L Maffei and L Mecacci, *La Visione dalla Neurofisiologia alla Psicologia,* A Mondadori, Milan (1979)
10 D C Van Essen, Functional Organization of the primate visual cortex, in *Cerebral Cortex,* Vol 3, *Visual Cortex* (A Peters and E G Jones, eds), pp 259–329, Plenum Press, New York (1985)
11 M Mishkin, L G Ungerleider, and K A Macko, Object vision and spatial vision two cortical pathways, *Trends Neurosci* **6,** 329–342 (1983)
12 P J Burt, C H Anderson, J O Sinniger, and G van der Wal, A pipeline pyramid machine, in *Pyramidal Systems for Computer Vision* (V Cantoni and S Levialdi, eds), pp 133–152, Springer-Verlag, Berlin (1986)
13 D Marr, *Vision,* Freeman, San Francisco (1982)
14 B Julesz, Early vision, focal attention, and neural nets, in *Neural Networks Theory and Applications* (R J Mammone and Y Zeevi, eds), pp 209–216, Academic Press, San Diego, CA (1991)
15 R Dodge, Five types of eyes movements in the horizontal plane of the field of regard, *Am J Physiol* **8,** 307–329 (1903)
16 C Rashbass, The relationship between saccadic and smooth tracking eye movements, *J Physiol* **159,** 326–338 (1961)
17 D A Robinson, Control of eye movements, in *Handbook of Physiology,* Section I, *The Nervous System* (V B Brooks, ed), Vol II, pert 2, pp 1275–1320, American Physiological Society, Bethesda, MD (1981)
18 D Sagi and B Julesz, "Where" and "what" in vision, *Science,* **228,** 1217–1219 (1985)
19 Th Wertheim, Peripheral visual acuity, *Am J Optom Physiol Optics* **57,** 915–924 (1980) [English translation of the original paper published in *Z Psychol Physiol Sinnensorg* **7,** 172–187 (1891)]
20 E L Schwartz, Computational anatomy and functional architecture of striate cortex a spatial mapping approach to perceptual coding, *Vision Res* **20,** 645 (1980)
21 P J Burt, Attention mechanisms for vision in a dynamic world, Proc 11th Int Conf on Pattern Recognition, Rome, I, pp 977–987 (1988)
22 C H Meyer, A G Lasker, and D A Robinson, The upper limit of human smooth pursuit velocity, *Vision Res* **25,** 561–563 (1985)

23. D. A. Robinson, Vestibular and optokinetic symbiosis, an example of explaining by modelling, in *Control of Gaze by Brain Stem Neurons* (R. Baker and A. Berthoz, eds.), pp. 49–58, Elsevier/North-Holland, Amsterdam (1977).
24. A. Buizza and R. Schmid, Visual-vestibular interaction in the control of eye movements: mathematical modelling and computer simulation, *Biol. Cybernet.* **47,** 203–211 (1982).
25. W. A. MacKay and J. T. Murphy, Cerebellar modulation of reflex gains, *Prog. Neurobiol.* **13,** 361–417 (1979).
26. J. Dichgans and Th. Brandt, Visual-vestibular interaction: effects on self-motion perception and postural control, in *Handbook of Sensory Physiology,* Vol. VIII, *Perception* (R. Held, H. Leibowitz, and H. L. Teuber, eds.), pp. 755–804, Springer-Verlag, Berlin (1978).
27. A. L. Yarbus, *Eye Movements and Vision,* Plenum Press, New York (1967).
28. D. Noton and L. Stark, Eye movements and visual perception, *Sci. Am.* **224** (6), 34–43 (1971).
29. H. Freeman, Shape description via the use of critical points, *Pattern Recognition* **10** (3), 159–166 (1978).
30. S. L. Tanimoto and A. Klinger (eds.), *Structured Computer Vision: Machine Perception Through Hierarchical Computation Structures,* Academic Press, New York (1980).
31. A. Rosenfeld (ed.), *Multiresolution Image Processing,* Springer-Verlag, Berlin (1984).
32. C. R. Dyer, Multiscale image understanding, in *Parallel Computer Vision* (L. Uhr, ed.), pp. 171–213, Academic Press, Orlando, FL (1987).
33. H. Samet, *Applications of Spatial Data Structures,* Addison-Wesley, Reading, MA (1991).
34. H. Samet, *The Design and Analysis of Spatial Data Structures,* Addison-Wesley, Reading, MA (1991).
35. H. Samet, The quad-tree and related hierarchical data structures, *ACM Comput. Surv.* **16,** 187–260 (1984).
36. A. Rosenfeld, Some techniques for image segmentation, in *Pyramidal Systems for Computer Vision* (V. Cantoni, and S. Levialdi, eds.), pp. 261–271, Springer-Verlag, Berlin (1986).
37. Ph. Clermont and A. Merigot, Efficient parallel pyramidal primitives or image analysis, in *Progress in Image Analysis and Processing.* II (V. Cantoni, M. Ferretti, S. Levialdi, R. Negrini, and R. Stefanelli, eds.), pp. 544–550, Wold Scientific, Singapore (1992).
38. V. Cantoni, L. Cinque, C. Guerra, S. Levialdi, and L. Lombardi, Describing object by a multi-resolution syntactic approach, Proc. 2nd Int. Conf. on Parallel Image Analysis, Ube, Japan, 1992.
39. P. J. Burt and E. H. Adelson, The Laplacian pyramid as a compact image code, *IEEE Trans. Commun.* **COM-31** (4), 532–540 (1983).
40. C. F. Neveu, C. Dyer, and R. T. Chin, Two-dimensional object recognition using multiresolution models, *Comput. Vision, Graphics, Image Process.* **34,** 52–65 (1986).
41. D. H. Ballard, Generalizing the Hough transform to detect arbitrary scapes, *Pattern Recognition* **13** (2), 111–122 (1981).
42. K. S. Fu, Recent developments in pattern recognition, *IEEE Trans. Comput.* **C-29** (10), 845–854 (1980).
43. K. S. Fu, Hybrid approaches to pattern recognition, NATO ASI on *Pattern Recognition: Theory and Applications* (J. Kittler, K. S. Fu, and L. S. Pau, eds.), pp. 139–155, D. Reidel, Dordrecht (1982).
44. N. Nilsson, *Problem Solving Methods in Artificial Intelligence,* McGraw-Hill, New York (1971).
45. R. S. Michalsky, Pattern recognition as rule-guided inductive inference, *IEEE Trans.* Pattern Anal. Machine Intell. **2** (4), 349–361 (1980).

46. W. H. Tsai and K. S. Fu, Attributed Grammar—a tool for combining syntactic and statistical approaches to pattern recognition, *IEEE Trans. Syst., Man Cybernet.* **SMC-10** (12), 873–884 (1980).
47. S. L. Tanimoto, J. P. Crettez, and J. C. Simon, Alternative hierarchies for cellular logic, Proc. 7th Int. Conf. on Pattern Recognition, Montreal, Canada, 1984, pp. 236–239.
48. P. J. Burt, Tree and pyramid structures for coding hexagonally sampled binary images, *Computer Graphics, Vision, Image Process.* **14,** 271–280 (1980).
49. N. Ahuja, On approaches to polygonal decomposition for hierarchical image representation, *Computer Graphics, Vision, Images Process.* **24,** 200–214 (1983).
50. N. P. Hartman and S. L. Tanimoto, A hexagonal pyramid data structure or image processing, *IEEE Trans. Syst., Man, Cybernet.* **SMC-14,** 247–255 (1984).
51. J. L. Crowley and A. C. Parker, A representation for shape based on peaks and ridges in the difference of low-pass transform, *IEEE Trans. Pattern Anal. Machine Intell.* **6** (2), 156–170 (1984).
52. W. G. Kropatsch, A pyramid that grows by power of two, *Pattern Recognition Lett.* **3** (9), 315–322 (1985).
53. S. G. Mallat, A theory for multiresolution signal decomposition: the wavelet representation, *IEEE Trans. Pattern Anal. Machine Intell.* **PAMI-11** (7), 674–693 (1989).
54. E. H. Adelson, E. Simoncelli, and R. Hingorani, Orthogonal pyramid transforms for image coding, in *Visual Communications and Image Processing.* II, *Vol. 845,* pp. 50–58 (1987). SPIE.
55. N. Millard and C. Paul, Recursive quadrature mirror filters: criteria specifications and design methods, *IEEE Trans. Acoust., Speech, Signal Process.* **ASSP-33** (4), 413–420 (1985).
56. W. M. Wells, Efficient synthesis of gaussian filters by cascaded uniform filters, *IEEE Trans. Pattern Anal. Machine Intell.* **PAMI-8** (2), 234–239 (1986).
57. J. Babaud, A. P. Witkin, M. Baudin, and R. O. Duda, Uniqueness of the Gaussian kernel for scale space filtering, *IEEE Trans. Pattern Anal. Machine Intell.* **PAMI-8** (1), 26–33 (1986).
58. A. L. Yuille and T. A. Poggio, Scaling theorems for zero crossings, *IEEE Trans. Pattern Anal. Machine Intell.* **PAMI-8** (2), 15–25 (1986).
59. D. Marr and E. C. Hildreth, Theory of edge detection, *Proc. R. Soc. London* **B-207,** 187–217 (1980).
60. P. J. Burt, Smart sensing in a pyramid vision machine, *Proc. IEEE* **76** (8), 1006–1014 (1988).
61. P. J. Burt and E. H. Adelson, A multi-resolution spline with application to image mosaics, *ACM Trans. Graphics* **2** (4), 217–236 (1983).
62. A. R. Hanson and E. M. Riseman, Segmentation of natural scenes, in *Computer Vision Systems* (A. R. Hanson and E. M. Riseman, eds.), pp. 129–174, Academic Press, New York (1978).
63. J. L. Crowley and R. M. Stern, Fast computation of the difference of low pass transform, *IEEE Trans. Pattern Anal. Machine Intell.* **6** (2), 212–222 (1984).
64. E. H. Adelson, C. H. Anderson, J. R. Bergen, P. J. Burt, and J. M. Ogden, Pyramid methods in image processing, *RCA Eng. Mag.* **29** (6), (1984).
65. A. Rosenfeld and M. Thurston, Edge and curve detection for visual scene analysis, *IEEE Trans. Comput.* **TC-20,** 562–569 (1971).
66. A. P. Witkin, Scale-space filtering, Proc. 7th Int. Joint Conf. IJCAI, 1983, pp. 1019–1021.
67. A. Haar, Aur theorie der orthogonalen functionensysteme, *Math. Ann.* **69,** 331–371 (1910).
68. J. E. Shore, A two dimensional haar-like transform, NLR Report 7472 AD 755433 (1973).
69. L. Carrioli, A pyramidal Haar transform implementation, in *Image Analysis and Process-*

ing (V. Cantoni, S. Levialdi, and G. Musso, eds.), pp. 99–108, Plenum Press, New York (1986).
70. M. Ferretti, Overlapping in compact pyramids, in *Pyramidal Systems for Computer Vision* (V. Cantoni and S. Levialdi, eds.), pp. 238–251, Springer-Verlag, Berlin (1986).
71. M. D. Levine, Region analysis using a pyramid data structure, in *Structural Computer Vision* (S. L. Tanimoto and A. Klinger, eds.), pp. 57–100, Academic Press, New York (1980).
72. G. Granlund and J. Arvidsson, The GOP image computer, in *Fundamentals in Computer Vision* (O. Faugeras, ed.), pp. 443–458, Cambridge University Press, Cambridge (1983).
73. M. D. Kelly, Edge detection in pictures by computer using planning, in *Machine Intelligence 6* (B. Meltzer and D. Michie, eds.), pp. 397–409, Edinburgh University Press, Edinburgh (1971).
74. P. J. Burt and G. van der Wal, An architecture for multi-resolution, focal, image analysis, Proc. 10th Int. Conf. on Pattern Recognition, Atlantic City, NJ, 1990, pp. 305–311.
75. C. H. Anderson, P. J. Burt, and G. van der Wal, Change detection and tracking using pyramid transform techniques, Proc. SPIE Conf. Intelligent Robots and Computer Vision, 1985, pp. 72–78.
76. V. Cantoni and S. Levialdi, Contour labelling by pyramidal processing, in *Intermediate-level Image Processing* (M. B. Duff, ed.), pp. 179–188, Academic Press, London (1986).

Chapter 3

Hierarchical Homogeneous Topologies

This chapter introduces a first class of hierarchical systems: the simplest way to assemble a hierarchy is to link identical modules, thus giving rise to homogeneous systems. Among the possible topologies that fit into this description are the following families: snowflakes (simple and dense), stars (partial and full), trees (regular, half- and full-ringed, hypertrees, multitrees, flip trees), hypernets, and pyramids (bin, quad, generic). Each family will be analyzed in detail. The analysis will be focused on topological aspects.

3.1. INTRODUCTION

There are different approaches when designing a system which has forms of massive parallelism, according to the autonomy of the computing unit and the capability of the interconnection network. Among the various taxonomies proposed, a basic distinction[1] can be drawn between systems which consist of full-functioning, autonomous processing elements (PEs), equipped with local memory, which have been called *multicomputer,* and systems that interconnect memory modules and computing modules, the *multiprocessors.*

The latter approach considers the interconnection network as a central resource of the system; further analysis has highlighted a possible partitioning of this class according to the communication mechanism between processing mod-

ules and memory modules. One can distinguish between loosely coupled systems, in which links are logical and communications can be assimilated with packet switching, and tightly coupled ones, where physical connections are established by means of memory sharing.

In the former approach, the network results from the interconnection of the PEs through dedicated links. Generally, such systems are analyzed under the assumption that the computations to be carried out are best described according to the multiple-instruction, multiple-data (MIMD) paradigm: i.e., autonomous, but cooperating tasks are executed on different PEs, exchanging data asynchronously via a message-passing technique. The different systems which have been proposed arise from the various topologies that have been used to exploit the limited number of links available with every processing element. Since the ultimate goal of these studies is the construction of systems containing a large number of units, the issues of modular design, regularity of decomposition, and fault tolerance are of the utmost importance.

Various topologic arrangements of PEs have been proposed and sometimes used in constructing *hierarchical multicomputer systems*. The different alternatives are here compared in terms of a few parameters describing the performance of the network (diameter, message traffic over the links, and so on) and the overall complexity of the system (the number of links as a function of the number of PEs). A few comments regarding addressing and routing algorithms, fault tolerance, VLSI implementation, and domain of applications will be reported as well.

3.2. HIERARCHICAL PARADIGM

Among the possible topologies for systems that share a homogeneous, modular, and regularly decomposable design, a paradigm can be defined as follows:

- A *node* consists of a single PE, equipped with local memory for program and data storage and with a small number of interconnecting channels.
- The degree d of a node is the number of connections with other nodes; each connection is bidirectional and can take the form of a dedicated link or of an access to a shared resource, such as a bus.
- A *module* is the elementary building block of the system, assembled with a set of p interconnected nodes.

- A *hierarchy* $H(n,p)$ results from a regular composition of modules arranged in such a way as to produce a structure with n levels.
- The total number of nodes in the structure will be denoted by N.

The overall structure resulting from the composition of modules can be enlarged modularly according to one of the two following approaches. Let $H(n,p)$ be a hierarchy of level n; $H(n+1,p)$ is the new structure obtained either by adding to $H(n,p)$ a proper number of modules to build the next level of the hierarchy or by replicating $H(n,p)$ and connecting these replicated subunits together. In both cases, no changes are required to $H(n,p)$ to enlarge the system, which is therefore, in terms of construction, uniform and regular (recursively decomposable).

The topologies that fit this description are the following families: snowflakes and stars,[2, 3] trees,[4–6] hypernets[7] and pyramids.[8, 9] Each family will be analyzed in detail in subsequent sections. It can be seen, however, that they can be thought of as belonging to two different larger groupings, according to the microarchitecture of the composing module; in snowflakes and stars (and in some cases in hypernets as well) the PEs are interconnected locally by a bus, while other families use a set of distinct physical channels to link the PEs of a module. The former group will be referred to subsequently as "bus-oriented architectures," the latter as "link-oriented architectures." This distinction could lead to different criteria evaluating the performance of the structure, as far as data exchanges are concerned, since one should take into account the contention for shared resources (the bus). For the purpose of this analysis, however, data transmission between immediately neighboring PEs will be considered as an atomic action; handshaking protocols, timing philosophy, and general control strategy will not be taken into account. The analysis will only be focused on topological aspects.

Regarding this, fault tolerance strategies will be considered simply from the point of view of alternative routes; the other elements of the fault tolerance implementation (namely, masking, detection, containment, diagnosis, repair–reconfiguration, and recovery[10]) for improving system reliability are outside the scope of this discussion.

3.3. COMPARISON PARAMETERS AND EVALUATION CRITERIA

To produce a sound quantitative estimate of the efficiency of such architectures, independent of their very specific application, it seems worthwhile to

introduce two types of measures. The former tries to quantify the complexity of the systems as generated by the topology alone: this leads to the introduction of the *increasing law* and *diameter* parameters. On the other hand, the latter approaches the problem of cost estimation for data exchanges among nodes in the system. This second class of parameters, which includes *loads, average internode distance,* and *message density,* depends heavily on a number of hypotheses on the computational model of tasks being executed on the system. The definitions of these parameters are given below.

As mentioned, a very general paradigm is the MIMD modality; the tasks are allocated to distinct nodes, possibly each node having a single task, and therefore the cooperative effort of the different computations puts a very heavy influence on the efficiency of data exchanges. To predict the behavior of the system it is necessary to model the process of message generation. The most generally used approach is based on the uniform reference model (URM). Within this model,[1] the probability $P\{i,j\}$ that node i sends a message to node j is constant for the whole system and does not depend on the distance between the two nodes. This assumption gives essentially an analytic simplification to all mathematical derivations of closed-form formulas for the parameters, but is rather crude, even for the MIMD situation. The introduction of a degree of locality in the interaction of cooperating tasks[2, 7] could take into consideration the characteristics of fine-grained parallel processes; in these cases, the probability $P\{x\}$ that two nodes at a distance x exchange messages has been modeled according to the *threshold* criterion[2] (each node has a local region of nodes that are addressed with equal probability; all other nodes outside such a region are uniformly addressed) or to the *geometric distribution* criterion[11] (nodes are grouped into regions, each region at increasing distance and geometrically decreasing probability of being visited by messages; within a region, all nodes are uniformly addressed). This latter kind of analysis has not been followed up here; as to the former some comments will be given, but the bulk of the chapter refers to the URM.

However, a further hypothesis has been made characterizing the behavior of these systems, namely that the message generation rate complies with the request independence assumption (RIA)[12]; in the context of multicomputer systems, this hypothesis states that each message generated at a node is independent from the previous one generated at the same node.

A few comments (and references to specific solutions) for the existence of detour paths will be given for the purpose of fault containment. Indeed the trade-off between the efficiency of fault-recovery algorithms and the complexity of the required hardware to implement them has to be individually evaluated for each realization.

For these assumptions, here are the definitions of the parameters that characterize the topology of the systems.

- *Increasing law* This parameter gives the overall number of links within the system as a function of the total number of nodes.

- *Diameter* Let a path length between nodes (i,j) be the number of nodes visited by a message traveling from node i to node j (including the destination node j itself) by atomic neighbor data movements. Let MinDist(i,j) be the length of the shortest path connecting two nodes i and j. Then

$$\text{Diameter} = \text{Max } \{\text{MinDist}(i,j)\} \quad \forall i,j \text{ in the system}$$

The following definitions are related to parameters that measure the performance of the system in message exchanges under the hypotheses of URM and RIA.

- *Average internode distance.* By weighting the number $N(x)$ of nodes at a distance x from a reference node with the distance itself and averaging on the number $N-1$ of the other nodes in the system, one obtains an average distance which depends on the reference node and that varies within the system if the structure is not regular. If this measure is computed for all possible choices of the reference node and again averaged, the resulting average internode distance is independent of the reference node and more fitted to describe the behavior of the system in terms of average cost in time for information exchange under URM.

- *Maximum load.* The load of a link is defined as the probability that a message between a random pair of nodes transits over that link. The location in the system, where it is at a maximum, is a possible bottleneck for data transmission.

- *Minimum load.* Opposite the previous parameter, the minimum load link shows the location in the system which is most underused.

- *Message density.* This is the average load in the whole system, obtained under the hypotheses that each node sends a message to all other nodes, resulting in a number of messages M where $M=N(N-1)$. The traffic in the systems is represented by the product of M with the average internode distance

averaged over the total number of links L. The density at each link is normalized with the number of messages, and the net result is that the message density is obtained as the ratio of the average internode distance to the number of links, a result to be expected under URM.

3.4. BUS-ORIENTED ARCHITECTURES

The common feature of bus-oriented architecture is the arrangement of PEs in the module; a bus interconnects the p nodes which make up the module. The family is composed of *snowflake* and *dense snowflake* structures and also of the *star* one; hypernets based on the buslet can also be considered here, but they are treated in a separate section under the more general family of hypernets. Each structure in this family is uniquely identified by the couple (n,p), n being the level of the hierarchy and p the cardinality of the module.

3.4.1. Snowflakes and Dense Snowflakes

An (n,p) snowflake[2, 3] is defined recursively as follows:

- A $(1,p)$ flake is the module.
- An (n,p) flake is composed of p flakes of level $n-1$, each having a node (called the active corner of level $n-1$) selected to connect to the bus of level n.

Within each $(n-1,p)$ flake, active corners must be in different positions; moreover, each flake (associated with a label r $(0 \leq r \leq p-1)$) has a so-called latent corner, to be used in building the $(n+1,p)$ flake, acting as the active corner of the (n,p) flake as shown in Figure 3.1. An example of a snowflake (3,4) is given in Figure 3.2.

A unique path from every source to every destination node can be defined, and therefore a simple (n,p) snowflake is not fault tolerant. The routing algorithm can be developed recursively as well: if the source and destination nodes belong to the same flake, the data are transmitted directly through the bus; conversely the data are transmitted from the source to the active corner and from this to the next active corner of the next higher level recursively until a level in which a subflake including the destination node is reached; through the active corners, once more recursively, data are transmitted downward through the subflakes until the destination flake is reached. As an example, Figure 3.3 shows the routing of a message from node 301 to node 121.

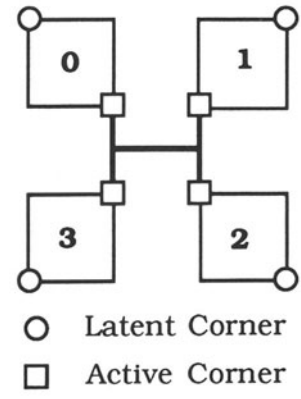


Figure 3.1. $(n,4)$ snowflake with the four components of level $n-1$. Note that the clockwise labeling sequence is used recursively throughout the levels.

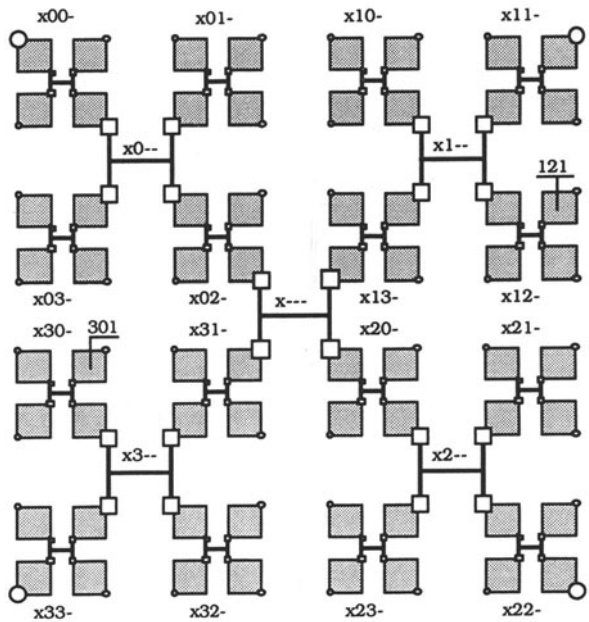


Figure 3.2. (3,4) snowflake: buses have an (n + 1)-digit label beginning with x and containing as many hyphens as the levels they belong to; the latent corners are 000, 111, 222, 333.

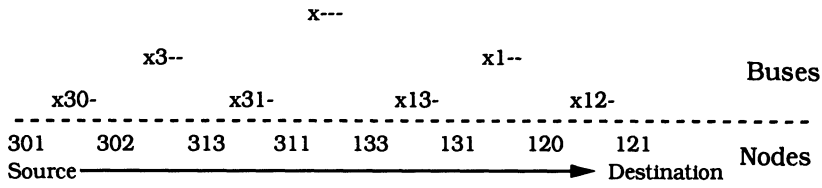


Figure 3.3. Example of the path between a pair of leaf nodes (301 → 121) in the (3,4) snowflake of Figure 3.2.

It is easily verified that the root bus must be traversed by most messages and therefore sustains the maximum load, while the minimum is that of the buses of the lowest level; this happens in most hierarchical systems.

An (n,p) dense snowflake[2, 3] is defined recursively as follows:

- A $(1,p)$ flake is the module.
- An (n,p) dense flake is composed of p dense flakes of level $n-1$, each connected to $p-1$ external buses of level n.

The active corners of each $(n-1,p)$ dense flake are therefore $p-1$ and access two buses; the single latent corner is connected only to the internal bus. An example of a (3,4) dense snowflake is shown in Figure 3.4.

In this kind of snowflake there exist more paths from a source to an arbitrary destination node. A "good" routing algorithm is the following (in Ref. 3 an optimum routing algorithm based on a different addressing scheme is presented): if the source and destination nodes belong to the same flake the data are transmitted directly; conversely, the message is transmitted from the source to the active corner of the nearest higher-level bus and from this to a next active corner of a higher level recursively until it reaches a level in which a subflake includes the destination. Through the active corners, once more recursively, the message is transmitted downward within the subflakes which include the destination until it reaches the level of the destination flake.

As an example, the path between the same node pair 301 and 121 of Figure 3 is shown in Figure 3.5: the message crosses only two intermediate nodes.

The introduction of more active corners at each level reduces by a factor of $p-1$ the load on the root bus, which remains, however, the maximum loaded one. Should this bus become unusable, alternative paths can be easily identified. Then, for alternative detours, each flake has $p-2$ additional message paths, providing a significant performance improvement to fault tolerance. Nevertheless, a node is disconnected when its two buses are faulty.

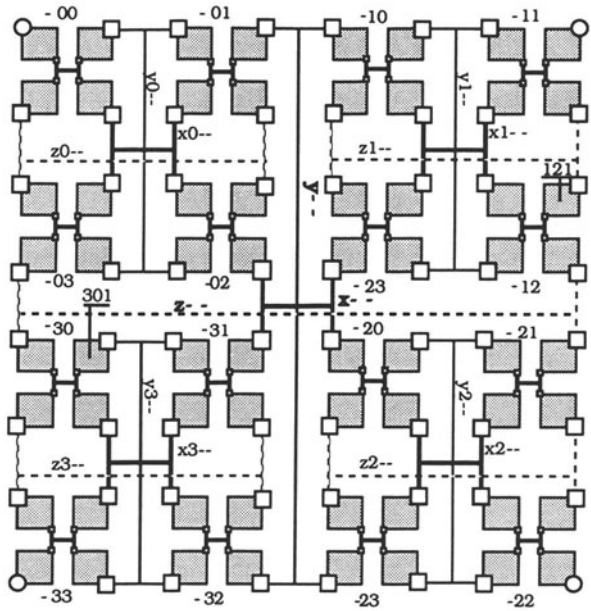


Figure 3.4. (3,4) Dense snowflake: bus labeling is the same of Figure 3.2; the three bus families are distinguished by the first letter, which is respectively *x*, *y*, *z*.

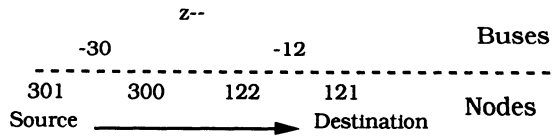


Figure 3.5. Example of the path between a pair of leaf nodes (301 → 121) in the (3,4) dense snowflake of Figure 3.4.

3.4.2. Partial and Full Stars

A partial and full stars structure[3] uses a module that consists of $p-1$ nodes connected to a bus; the hierarchy results from the connection of $p-1$ modules to the new higher-level one via the lower-level buses, and not through lower-level nodes, chosen as active ones, as is the case in snowflakes family. All in all each bus, except the root one, hosts p nodes.

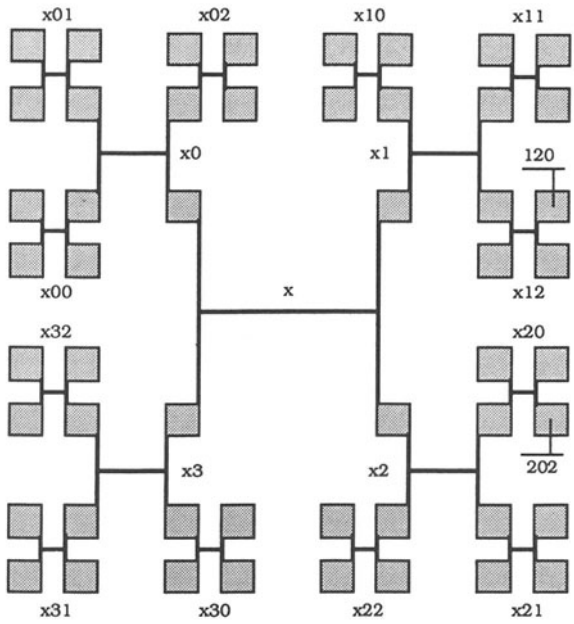


Figure 3.6. (3,4) full star: bus labeling follows the same addressing rule of the nodes; the number of digits corresponds to the level of the bus.

An (n,p) star is defined recursively as follows:

- A $(1,p)$ star has $p-1$ nodes connected to a single bus.
- An (n,p) star uses a new module with $p-1$ nodes, each connected to the highest bus of a different $(n-1,p)$ substar.

This definition generates the *partial star;* indeed the top-level bus can host a last node, which can be used to complete the structure, thus obtaining the *full star*. In this case the whole hierarchical structure can be seen as an alternate sequence of buses and processors; every bus connects one node of a higher level module with $p-1$ nodes of a lower level one. The center of the cluster is the top-level bus, while the periphery is made up of the leaf nodes.

The addressing rule is such that every node is assigned a variable-length address where the least significant digit (ranging from 1 to $p-1$) is related to the top-level nodes and the other digits are added to the left in order to address lower-level nodes. Figure 3.6 shows a (3,4) full star.

As in the snowflake family, a unique path can be found for every pair of

source–destination nodes. The routing algorithm is also simple: starting from the source node the message is transmitted from lower-level bus to higher-level bus recursively until a common "parent" between the substar of the source and that of the destination node is reached. Then the message is transmitted downward within the substars until it arrives at the destination.

As an example the path between the pairs of nodes 202, 120 represented in Figure 3.6 will be $202-x20-20-x2-2-x-1-x1-12-x12-120$.

The topology of the full star is closely related to that of the snowflake; indeed, routing algorithms and maximally and minimally loaded buses coincide. Obviously, since there is a unique path from every source to every destination, no detour technique for fault purposes can be conceived.

3.5. LINK-ORIENTED ARCHITECTURES

The modular construction approach which augments the levels of the hierarchy by replicating subunits, as previously seen for the bus-oriented architectures, is replaced in the link-oriented ones by an incremental strategy no longer based on the module but relying directly on the nodes. *Regular trees* are the basic structure of the family. However, additional links are required to provide alternative paths which improve the performances of the network and are the basis for achieving some fault tolerance in message routing. Cases considered are *ringed trees* and *hypertrees, multitrees, fliptrees,* and *bin, quad,* and *generalized pyramids*. This broad group extends the hierarchical family based on trees to the topology of graphs. To this family also belong *hypernets,* even if some instances of these architectures embody buses.

3.5.1. Regular Trees

A regular m-ary tree of n hierarchical levels is defined recursively as follows:

- A $(1,m)$ tree consists of a single node, called the root of the tree.
- An (n,m) tree is composed of a root and of m disjoint sets of nodes, each forming an $(n-1,m)$ tree of level $n-1$.

The degree d of the nodes varies within the structure; $d=m+1$ except for the root which has $d=m$ and for the leaves where $d=1$.

A symmetric visit of the tree generates the following numbering scheme for the nodes: the root is labeled 1; the m children of a node labeled x receive

labels $mx+i$, $0<i<m-1$. These labels, in base m, give each node a variable-length address; the number of digits in the address is equal to the position of the node in the hierarchy. Figure 3.7 shows a (5,2) tree.

With this definition, the routing algorithm is quite simple: the source and destination node belong to a common subtree (possibly the whole tree) identified by a number of common most significant digits in their addresses. A message has to move upward to the root of the common subtree and to descend using the other digits of the destination address in order to move among the other subtrees until the destination node is reached.

Trees have unique paths between any two nodes and this leads to congestion problems at the higher levels of the hierarchy and to critical fault tolerance problems. Each link of the root node supports a load of roughly $2(m-1)/m^2$; the minimum load is found at the leaf nodes and is approximately $2/N$.

Various mechanisms for implementing fault-tolerant reconfigurable trees have been proposed in recent years. The first proposals in the early 1980s aimed at the construction of tolerant trees for one and two faults.[13, 14] In Ref. 15 these proposals have been improved by adding sufficient redundant links in order to tolerate multiple failures. A new approach, in which spare nodes are allocated at the leaves, has been followed in SOFT (subtree-oriented fault toler-

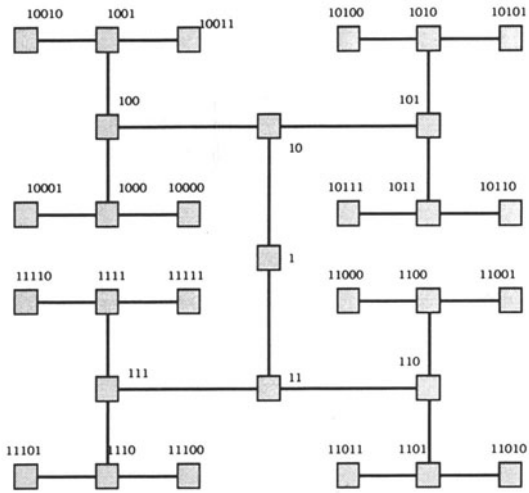


Figure 3.7. (5,2) regular binary tree. In this planar embedding each child of a node receives the label of the father augmented with 0 for the left child and 1 for the right child.

ance)[16]; it achieves a considerable flexibility in reconfiguration, with significantly less redundancy, through sharing of spares between adjacent subtrees.

3.5.2 Augmented Trees

Additional links to the skeleton tree can be used in order to reduce the average path length and to provide a more uniform message density. The placement of the additional links so far proposed leads to half- and full-ringed trees and, when the average distance between nodes is minimized, to the hypertree. Obviously, the addressing scheme for these augmented structures is the same as the one for regular trees.

3.5.2.1. Full- and Half-Ringed Trees

The class of structures[4] containing full- and half-ringed trees augments the connectivity of regular trees by adding at most two links to every node apart from the root. The full-ringed tree has exactly two such extra links per node that generate at each level a ring; the half-ringed tree only interconnects, with a single extra link, neighboring nodes that are not siblings. Figures 3.8 and 3.9 show respectively a (5,2) full-ringed tree and a (5,2) half-ringed tree.

To define a routing scheme for the full-ringed tree, let $D(i,j)$ be the distance along the ring of two nodes i and j which belong to the same level. If the source and destination nodes are at the same level in the hierarchy, the routing algorithm chooses the horizontal connections along the ring if

$$D(i,j) \leq D(\lfloor i/m \rfloor, \lfloor j/m \rfloor) + 2 \tag{3.1}$$

(simulations show that replacing $\leq$ by $<$ leads to a less uniform load distribution). Vertical hops are handled in the same way as for regular trees.

A routing algorithm for the half-ringed binary tree case can be found in Ref. 17.

The congestion of the root links typical of regular trees is avoided in these structures. For example, communications among leaves in the tree never involve the root; the bottleneck is found at the links of the fourth hierarchical level (''horizontal'' links for the full-ringed and vertical links for the half-ringed case, respectively).

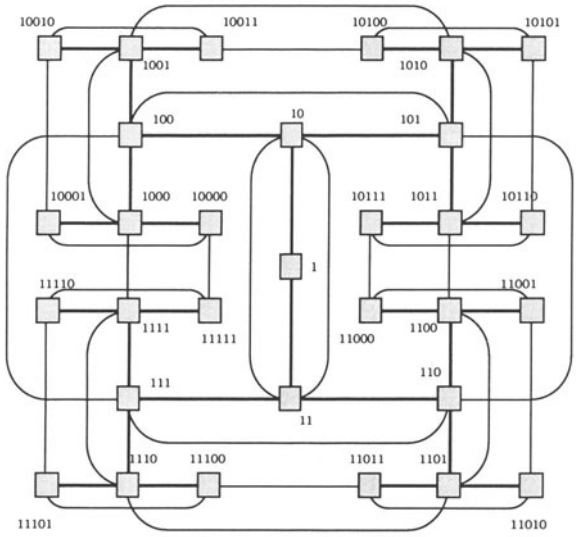


Figure 3.8. (5,2) full-ringed binary tree. Two extra links are added to each node, and a complete ring is realized at each hierarchical level.

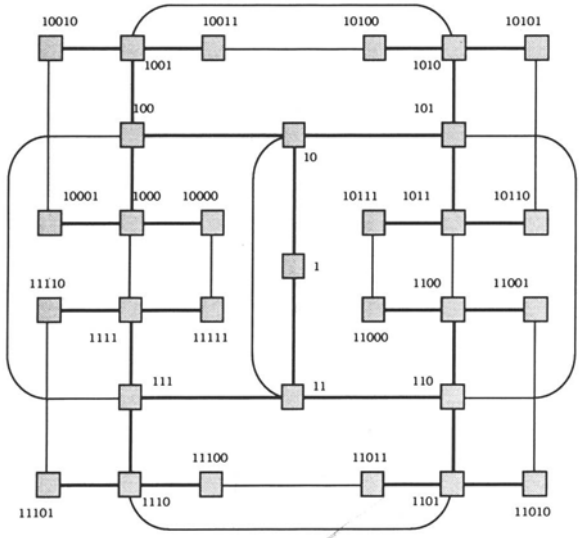


Figure 3.9. (5,2) half-ringed binary tree: with respect to the full-ringed binary tree the extra links between siblings are missing.

3.5.2.2. Hypertrees

Internode distances can be lowered by modifying the structure of a regular tree. A better utilization of extra links within a level should favor n-cube links—that is, links between nodes which have a Hamming distance of 1.

Such is the case in hypertrees.[4] According to the number of extra links introduced, one obtains a hypertree I or a hypertree II, and so on. Since at each level x of the hierarchy there is an n-cube structure of degree $x-1$, one has to choose which links to introduce (let us consider only hypertree I). Such extra links are chosen so that the maximum distance among the isolevel nodes becomes minimal.[4] A (5,2) hypertree I is shown in Figure 3.10.

As for the routing algorithm, it follows from the definition that a message uses a horizontal connection if it decreases the Hamming distance between the current position in the tree and the destination node. No path between leaf nodes goes higher than the middle level of the tree.[4] For this reason, the average distance of hypertrees is much better than those of all the other augmented trees.

The simple routing algorithms described can easily be updated to support the presence of no more than one faulty link or node, provided that the faulty

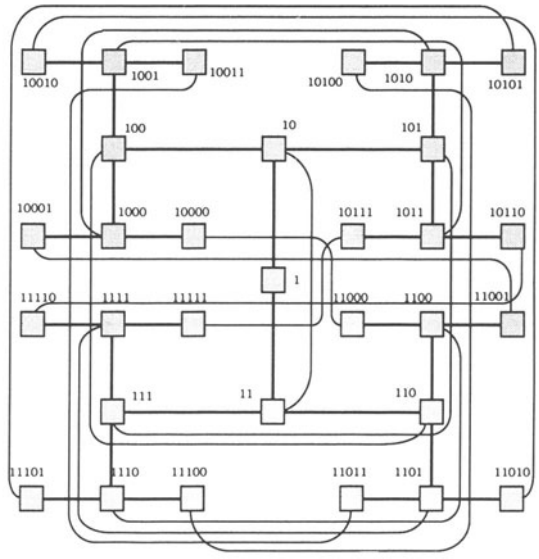


Figure 3.10. (5,2) hypertree I. As in the half-ringed binary tree one extra link for each node is added, in this case minimizing the maximum distance among the nodes in each layer.

node is not itself the source or target. In fact, each node has a minimum of one alternative path, and the detour can be limited to a few recovery steps toward the original path.[4]

While the "long-distance" n-cube interconnections are useful for the stated network performances, the lack of redundant paths to nearby nodes, such as those of ringed trees, may be more disadvantageous if the message traffic is mainly local and for generating the quoted short local recovery detour.

3.5.3. Complete Trees

The tree networks so far introduced are characterized by a different degree of node: usually the root and the leaves have unexploited connections available. Some proposals have been made to fully exploit the connection capabilities of the nodes in order to improve mainly the resilience to faults and the network performances. We consider the multitrees and the fliptrees. While the other tree families can be enlarged with a new level simply by adding new nodes and connections, for complete trees the updating is more complex: it is necessary to first remove the extra connections between the leaves and then proceed to the construction of the new level.

3.5.3.1. Multitrees

A multitree[5] is a graph characterized by three parameters (n,d,t) with the following properties:

- A set of t identical component (n,d) trees defined as follows:
  The root of a component has $d-2$ children.
  All other nodes, except the leaves, have $d-1$ children.
- The roots of the component trees form a ring.
- Each leaf of the multitree has $d-1$ links with other leaves; all level-n nodes are linked in at least one cycle.

It follows from these definitions that the degree of all the nodes is constant in the structure and is equal to d. So, an (n,d,t) multitree has $N=t(d-1)^{n-1}$ nodes and $dN/2$ links. Figure 3.11 shows a (4,3,4) multitree.

A node in a multitree is labeled by a pair of numbers (x,y); x $(0 \le x \le t-1)$ indicates the component tree, and y is the address of the node in such a tree using a straightforward modification of the rules defined previously for regular trees.

No routing algorithms has yet been given for this structure, which is so

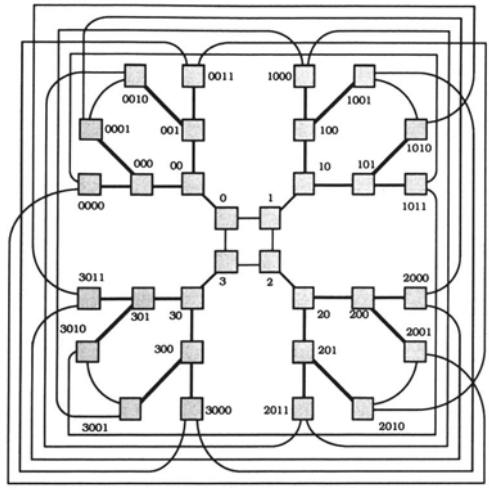


Figure 3.11. (4,4,3) multitree. Fully exploiting the d-connectivity of each node, extra rings are introduced for apexes and leaves respectively.

rich in alternative paths that it is tolerant of $d-1$ faults; empiric results can be found in Refs. 5 and 18.

3.5.3.2. Flip trees

An (n,d) flip tree[6] is defined as follows:

- A set of d regular $(n-1,d-1)$ trees.
- The root of each $(n-1,d-1)$ subtree has a link connected to the root node of the flip tree.
- Each leaf node has $d-1$ additional links with other leaf nodes, each one belonging to a different subtree.

The degree of a node is constant and is rigorously d, as in multitrees. A (4,3) flip tree is shown in Figure 3.12.

The addressing scheme assigns to the nodes an increasing number of digits; starting from the d children of the root, which receive labels 0,1, . . . , $d-1$. Lower-level nodes add a digit in the range $0,d-2$ to the address of the parent. The additional links at the leaves are identified as follows: by flipping the $n-2$ least significant digits of a leaf note address, one selects a set of $d-1$ leaf nodes each located in one of the remaining $d-1$ subtrees.

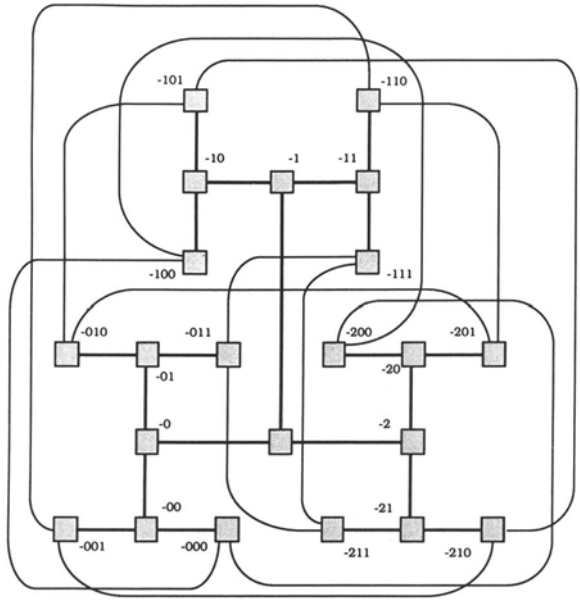


Figure 3.12. (4,3) flip tree. The d-connectivity of each node is used to achieve the highest fault tolerance resilience.

An explicit routing algorithm is given in Ref. 6. The additional links, with reference to the regular tree, are introduced mainly to support fault tolerance strategies. Indeed, the congestion remains higher than in the hypertree structure, but the flip tree is strongly resilient to faults: in the presence of a maximum of $d-1$ faults, the diameter increases to a maximum of $2n-1$ (a value larger by 2 than in the case of absence of faults). Furthermore, there exists a container (defined as a set of node-disjoint paths between a given pair of nodes) consisting of d paths (obviously of maximum length $2n-1$) between every pair of nodes.

3.5.4. Pyramids

A pyramid is the extension of a regular tree with a set of gridlike connections among the nodes of each level. Such an extension ties this structure to a bidimensional support and has been motivated mainly by applications such as image processing and computer vision.[19, 20]

Formally, four parameters are necessary to completely describe a pyramid:

1. The number of levels of the hierarchy (n).
2. The tessellation topology (t), which defines the connectivity among the nodes of the same hierarchical level. These nodes are distributed in a regular two-dimensional grid: each node can be connected only to the four immediate neighbors in the cardinal directions (4-connectivity); to the eight near neighbors, adding the four diagonal directions (8-connectivity); to six neighbors in a hexagonal tessellation (6-connectivity).
3. The width of the support (w) used to build each new level defines the interconnections between nodes which belong to adjacent hierarchical levels: each node has w children in the successive level of the hierarchy.
4. The degree of reduction between successive levels (r). If $r=w$ the pyramid has no overlapping: the sets of children of the nodes of each level are disjointed, no node belongs to two parents; in this case, the degree of intermediate nodes is $d=1+t+r$. For the overlapped case, $w>r$, and the maximum degree is $d=w+\lceil w/r \rceil+t$.

The analysis carried out here is limited to the nonoverlapped, square tessellated case ($r=w$; $t=4$).

An (n,w) pyramid of n hierarchical levels is defined recursively as follows:

- A $(1,w)$ pyramid consists of a single node, called the apex.
- An (n,w) pyramid consists of w $(n-1,w)$ pyramids interconnected by extra links among external isolevel nodes and by w vertical links from the $(n-1,w)$ apexes to the (n,w) apex.

As an example, Figure 3.13 shows the $(n,4)$ pyramid and its construction based on the $(n-1,4)$ subpyramids. The number of extra links in this case is $4(2^{n-1}-1)$.

Within nonoverlapped pyramids, two cases shall be considered: the bin pyramid[8] ($w=2$) and the quad pyramid[9] ($w=4$). The former is a slightly anisotropic structure, since each parent has two children and, for alternate planes of the pyramid, vertical connections take place once along the rows and once along the columns respectively. This organization results in the casting of a binary tree into the vertical interplane connectivity. Conversely, the quad pyramid is an isotropic structure. Figures 3.14a,b show respectively a five-level bin pyramid and a three-level quad pyramid.

Several addressing algorithms can be adopted in a pyramid. Here the following solutions for bin and quad pyramids are suggested: the apex is labeled $(1,1)$; the children of a node (i,j) are labeled $(2i,2j)$, $(2i+1,2j)$, $(2i,2j+1)$, $(2i+1,2j+1)$ respectively in the quad pyramid and, in the bin pyramid, $(2i,j)$,

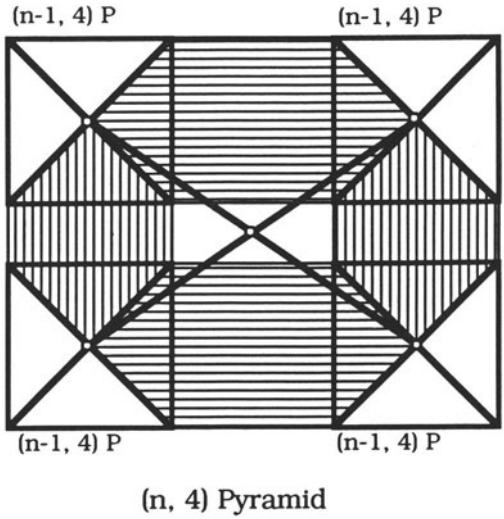


Figure 3.13. An $(n,4)$ pyramid with the four components of level $n-1$. Light lines show the links among isolevel nodes connecting the $(n-1,4)$ subpyramids; heavy lines are hierarchical links from the $(n-1,4)$ apexes to the $(n,4)$ apex.

$(2i+1,j)$ for even layers and $(i,2j)$, $(i,2j+1)$ for odd layers. The parent label is $(\lfloor i/2 \rfloor, \lfloor j/2 \rfloor)$ in the quad case and $(i, \lfloor j/2 \rfloor)$ for even layers and $(\lfloor i/2 \rfloor, j)$ for odd ones in the bin case. With the adopted addressing algorithm, the layer of a given node (i,j) is given by $\lfloor \log_2 i+1 \rfloor$ in the quad pyramid and $\lfloor \log_2 i \rfloor + \lfloor \log_2 j \rfloor + 1$ in the bin one.

The routing algorithm can easily be defined. Let $S=(i,j)$ and $D=(r,s)$ by the source and destination nodes, respectively, and let us suppose that the nodes belong to the same layer (otherwise, an extra path must be included in which data are sent from the node of the lower layer recursively upward until the layer of the higher node is reached). "Horizontal" ("vertical") communications are selected if the following relationships hold (do not hold):

$$\text{(quad)} \quad \text{abs}[i-r]+\text{abs}[j-s] \leq \text{abs}\lfloor {}^{i}/_{2} - {}^{r}/_{2} \rfloor + \text{abs}\lfloor {}^{j}/_{2} - {}^{s}/_{2} \rfloor + 2 \tag{3.2}$$

$$\text{(bin,even)} \quad \text{abs}[i-r]+\text{abs}[j-s] \leq \text{abs}\lfloor {}^{i}/_{2} - {}^{r}/_{2} \rfloor + \text{abs}[j-s] + 2 \tag{3.3}$$

$$\text{(bin,odd)} \quad \text{abs}[i-r]+\text{abs}[j-s] \leq \text{abs}[i-r] + \text{abs}\lfloor {}^{j}/_{2} - {}^{s}/_{2} \rfloor + 2 \tag{3.4}$$

This routing algorithm minimizes path length. Still, communications between remote nodes tend to produce congestion in upper layers, even if the apex itself is involved only if it corresponds to S or D.

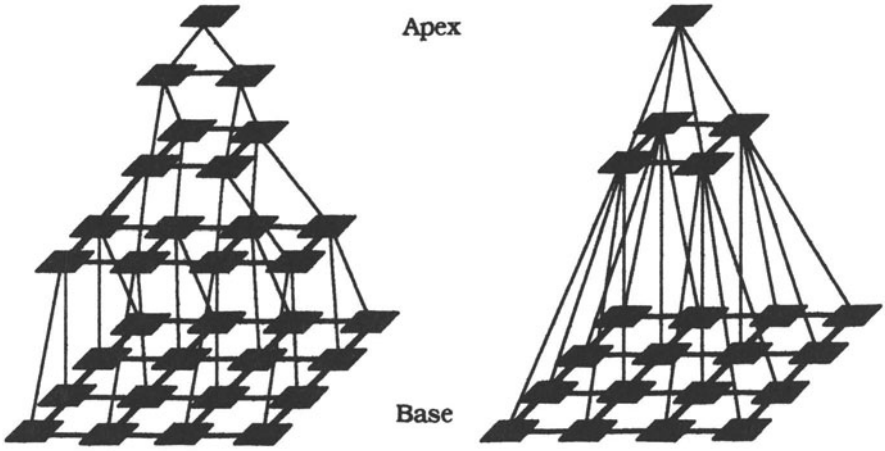


Figure 3.14. Binary and quad pyramids: (5,2) and (3,4) respectively with square tessellation on the layers.

In both the bin or quad cases the connections with nearby nodes are so rich that local, very short detours can be always realized when faults occur.

3.5.5. Hypernets

A hypernet hierarchical structure[7] combines the complementary characteristics of trees and hypercubes in the same system (modularity and low number of links in trees, strong connectivity, regularity, and symmetry in hypercubes), meanwhile avoiding their respective shortcomings (message congestion close to the root and weak fault tolerance in the first ones and unmodularity in the other ones).

A general hypernet is characterized by a quadruple (B,n,d,G) with the following definitions:

- B is the set of modules used for constructing the net.
- n is the number of levels of the hierarchy.
- d determines the number of external links (equal to 2^d) in each module.
- G is the global connectivity of the net or the number of interconnections between any pair of subnets.

A particular choice for B and G defines a certain family of hypernets, and n and d specify an instance inside this family. Usually the basic modules in the

whole structure are identical; in Figure 3.15 some examples of these modules are shown: cubelet, buslet, and treelet.[7]

Under these assumptions an (n,d) hypernet ($n \geq 1$) is defined recursively as follows:

- A $(1,d)$ hypernet consists of a single module with 2^d external links.
- An (n,d) hypernet is composed of $2^{2^n-2(d-2)+1}/G$ subnets, which are $(n-1,d)$ hypernets, each one interconnected to all others by G different external links selected among the unallocated ones; moreover in each sublet, G unused external links are dedicated to I/O channels.

As an example, in the construction of the $(2,d)$ hypernet, it can be verified that exactly one half of the links of every module are used; to be precise, G links are devoted to I/O, $2^{d-1}-G$ to the connections with the other modules (the subnets), while the remaining 2^{d-1} ones are still available for further connections. Figure 3.16 shows a (3,2) hypernet based on a treelet.

The addressing scheme divides the $m=\log_2 N$ bits (N being the total number of nodes in the net) that identify a node into n different fields: the field farthest left provides the address of the $(n-1,d)$ subnet containing the node, and so on, recursively to the field farthest right of d bits, which is the address of the node inside the basic module.

The connection rule adopted in constructing the structure is such that two different nodes are connected at level h via their external links if and only if the least significant $h-1$ digits in both addresses are identical to the binary

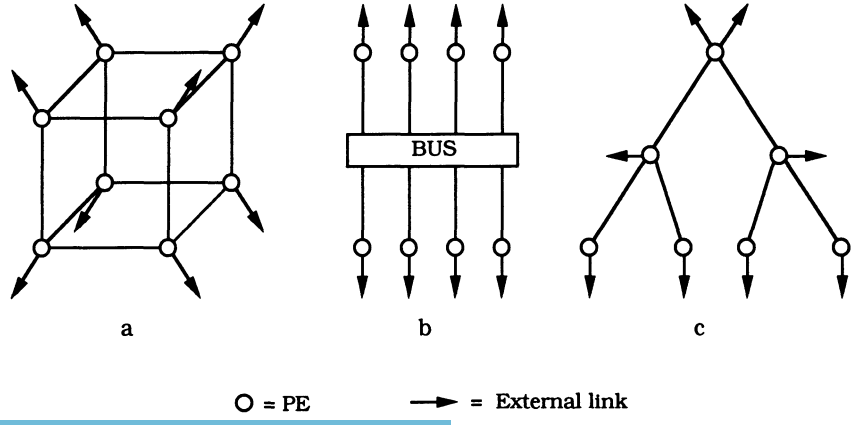


Figure 3.15. The three main modules of hypernets: (a) cubelet, (b) buslet, (c) treelet.

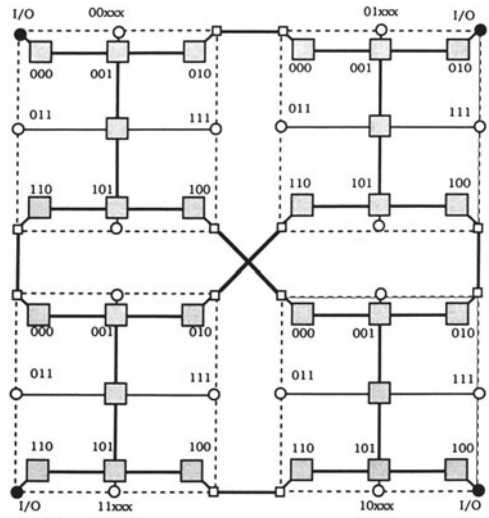


Figure 3.16. (3,2) hypernet based on treelet. Each node has five-digit address: the most significant two bits select the treelet modules; the remaining three select the nodes inside the modules. Note that the I/O nodes are identified by the black circles.

sequence 011 . . . 1 and each of the two addresses can be obtained from the other one by swapping the most significant q bits with the next significant group of q bits, where $q=2^{n-2}(d-2)+1$. Nodes having self-matching addresses in applying this second rule act as I/O nodes, and their external links provide I/O channels.

The message-routing scheme exploits the hierarchical nature of the structure; so if source and destination nodes are within the same basic module, routing is determined by its topology. Conversely, the path first proceeds up to reach the lowest subnet, including both nodes. It then runs down through lower subnets till the destination basic module is obtained.

The diameter and the average internode distance of an (n,d) hypernet depend on the nature of the building blocks: for an n-level hypernet these parameters are bounded by $2^{n-1}(d\{1\}+1)-1$, where $d\{1\}$ indicates the diameter of the basic module. Obviously, the maximum load is supported by the top-level links among the subnets of level $n-1$.

The minimum path is unique, although many other longer routes can be found. Without using external links for rerouting, the fault tolerance for the internal link failures of a module is possible only in the cubelet case (any faulty link can be sidestepped by two extra hops)[21]: in the other referenced cases there is only one path. But the (n,d) hypernet is based on a complete graph of $(n-1,d)$ subnets: the external links provide a container (G) so rich that the

destination node can be reached, for a single fault (both internal or external) in a few extra steps. For more details see Ref. 7.

3.6. PERFORMANCE MEASURES

The hierarchical architectures briefly described in Sections 3.4 and 3.5 can be compared[22] by using some of the performance indicators introduced in Section 3.3. The results of the comparison are shown in Figures 3.17–3.20; they contain a plot of the chosen parameters as a function of the system size—that is, the number N of nodes. Since single instances within each family behave differently, each family is characterized by a "band" showing the lowest and highest performances.

The "increasing law" has been selected among the parameters that depend only on the topology of the system and has been obtained by mathematical derivations in closed-form formulas. Figure 3.3 shows a plot of this indicator

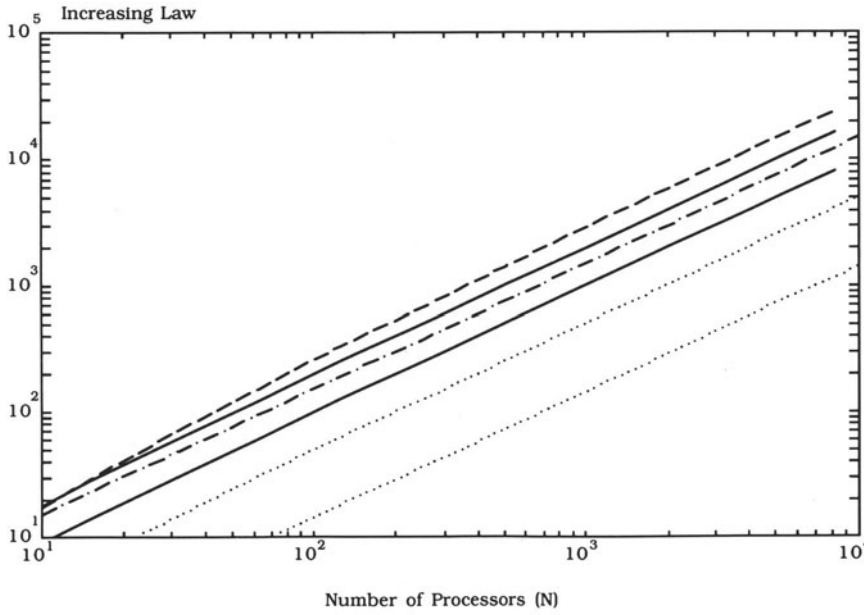


Figure 3.17. The increasing law number of links as a function of the number of nodes Bus architectures band within dotted lines ( ) Trees band within stroked lines (—) Pyramids dashed lines (--) Hypernets dot–dashed line (–)

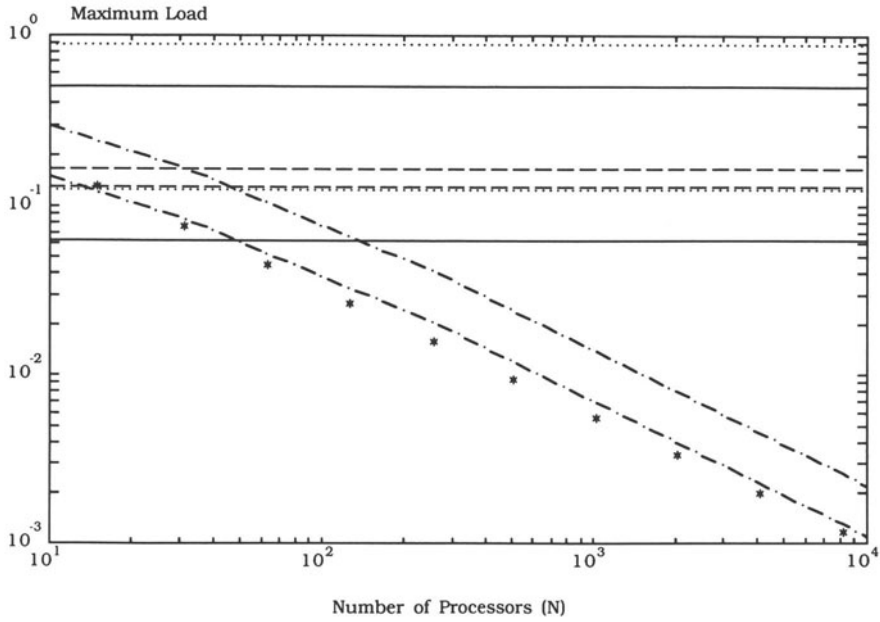


Figure 3.18. Maximum load as a function of the number of nodes. Bus architectures: band within dotted lines (···). Trees: band within stroked lines (—). Pyramids: band within dashed lines (--). Hypertrees: line of asterisks (*). Hypernets: band within dot–dashed lines (–·).

against the number of nodes in the system. The figure highlights the bands into which the families divide; the bus architectures rank best, but the figure does not take into account resource sharing among the module nodes. The band is identified by a lower limit (sparse snowflakes with $p=8$) and a higher one (dense snowflakes with $p=4$). The next band is associated with the family of trees ($d=3$); the lower limit is represented by the regular tree, while the higher one is the full-ringed tree. Hypernet systems show a very narrow band that collapses into a single line, located midway in the tree family; the highest line represents the family of pyramids. Almost by definition, the increasing law for these hierarchical systems is a linear function of the number of nodes; the proportionality factor is associated with d for link architectures and with $1/p$ for bus-oriented ones.

Of the three definitions of loads, the ''maximum load'' parameter has been used to compare the systems under the specified hypotheses of traffic models (for the tree and pyramid families the estimate is based on computer simulation;

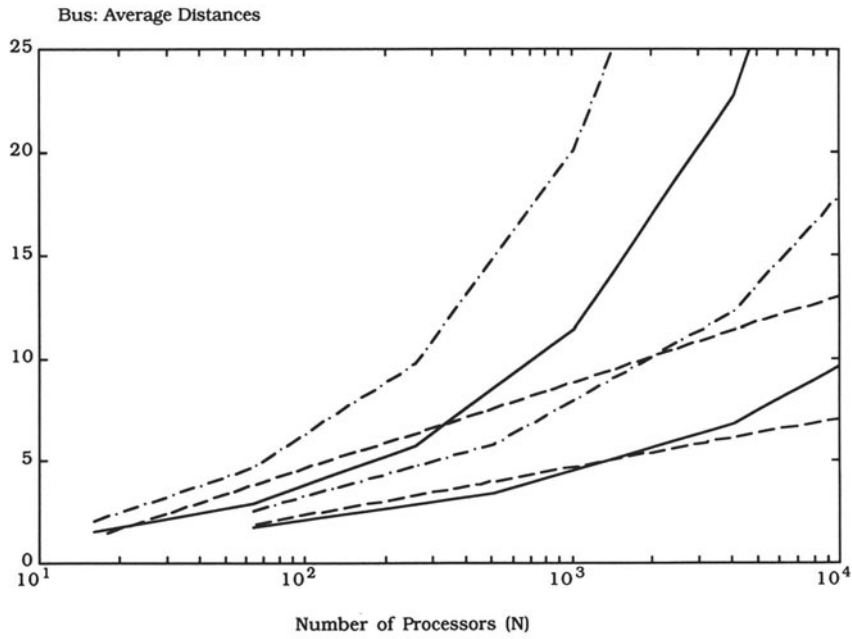


Figure 3.19. Average internode distance in bus-oriented architectures as a function of the number of nodes. Stars: band within dashed lines (--). Snowflakes: band within dot-dashed lines (–·). Dense snowflakes: band within stroked lines (___).

in the other cases a closed-form formula has been derived). Generally speaking, for a given N, the higher is p (or d), the lower is the number of levels n and, for link-oriented structures, the maximum load. As shown in Figure 3.18 the families of bus-oriented systems, pyramids, and trees all share a common behavior pattern: the maximum load is almost independent of the number of nodes. Indeed, these systems are constructed by increasing a level-$(n-1)$ hierarchy by adding another layer at the bottom, thus preserving the upper levels of the subsystem. Therefore, the maximum load, which is always sustained by these levels, remains fixed as the system grows. The only exception to this rule among trees is hypertrees: messages between leaves never reach higher than half the height of the tree, and, by augmenting the number of levels, each node in a layer can use links connecting it to others over longer distances. The load thus decreases as the number of nodes increases. In a similar way, in hypernets the number of links available to distribute the traffic increases as the structure grows: in fact all subnets are linked in the process of constructing the larger

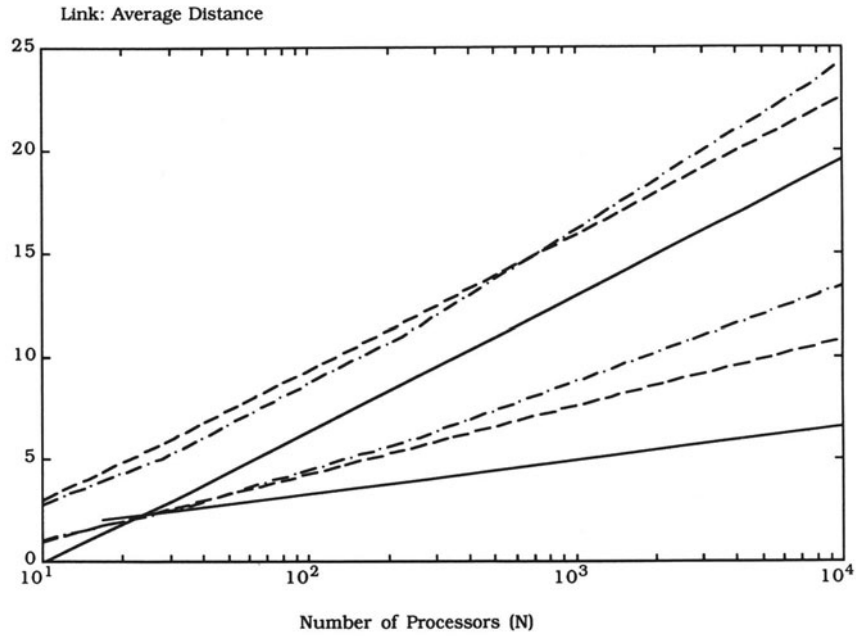


Figure 3.20. Average internode distance in link-oriented architectures as a function of the number of nodes. Trees: band within stroked lines (—). Hypernets: band within dot–dashed lines (–·). Pyramids: band within dashed lines (--).

system, and an increased number of subnets is used at each new level of the hierarchy.

The ''average internode distance'' is a second parameter selected to compare the systems with regard to message distribution and traffic congestion; the ''message density'' parameter is strictly tied to the average internode distance, as previously explained, so the analysis sheds light on both types of behavior. It is well known[5] that graphs of degree d having N nodes show a lower bound k on the diameter (known as the Moore bound) and, as a consequence, on the average internode distance too: such a bound is roughly $\log(N)/\log(d-1)$. Graphs reaching such a bound are called Moore graphs: an example is the (3,3) flip tree. As a rule of thumb, Moore graphs have an average internode distance equal to the number of levels in the embedded hierarchy, general graphs show a value roughly twice as large, a few instances of complete trees have an intermediate behavior; the worst case is in snowflakes, where the average internode distance is exponential in the number of levels.

Figure 3.19 contains the plots of the average internode distance as a function of the number N of nodes for the bus-oriented family (obtained in closed-form formulas), whereas Figure 3.20 contains those of the link-oriented systems (obtained in closed-form formulas but for the pyramids that have been simulated). Of the three bus systems, the star exhibits the best performance, with a logarithmic dependence on N. The snowflakes, instead, have larger internode distances because the connection between a generic pair of nodes most probably uses lower-level nodes, while in the full-star case the same connection would exploit buses located at higher hierarchical levels. The performances of the link-oriented families are very similar; their bands substantially overlap. The overall behavior shows a logarithmic increase in the number of nodes; the differences within a family strongly depend on the degree of the node.

The two parameters which estimate the cost of data exchanges, namely the maximum load and the average internode distance, have been evaluated in the tree and pyramid cases for data exchanges among leaf nodes.

The foregoing analysis has been carried out under the URM assumption. However, since many of the systems here discussed have been proposed also for application domains where the message traffic among nodes is rather local (such as vision), it is worthwhile making a few observations on the "threshold model."[2] Let us consider, for example, the average internode distance and define a "localized average distance"[7] d_{tm} under the threshold model as follows:

$$d_{tm} = \alpha d_1 + (1 - \alpha) d_{URM} \tag{3.5}$$

where d_{URM} is the average distance in the URM, d_1 is the average distance among the nodes located within the region defined by the threshold (it is a purely topological parameter), and α is the percentage of local traffic. The asymptotic behavior resulting from this formula when the traffic tends to be predominantly local ($\alpha \to 1$) is independent of the architecture of the system; little influence is played by the hierarchical structure because message routing within the region only partially exploits the hierarchy. On the other hand, the larger the region, the closer the behavior to the URM performance. A similar observation can also be made for the load; the maximum load moves from the higher links within the hierarchy to the lower ones. Generally, the performances of hierarchical structures under the threshold model are similar to those of the URM, provided that the dimensions of the system are large enough: in fact, αd_1 is a constant while $(1 - \alpha) d_{URM}$ depends on N.

3.7. APPLICABILITY

To appraise the usefulness of the hierarchical architectures so far examined, it is worth addressing at least two questions which are extremely important from the application point of view: the feasibility of the structures in terms of very large scale integration (VLSI) and the set of algorithms that can easily be mapped onto each family. Moreover, a brief review of the systems that have been built (or merely proposed) according to the outlined paradigms helps us understand the actual interest in the various architectures. The discussion does not cover pyramids, which are considered in detail in other chapters.

3.7.1. VLSI Feasibility

All the architectures reviewed here have been introduced as a means of building large systems, with the minimum number of processors being well into the hundreds. However, we can easily speculate that the bus-oriented family is liable to give rise to systems with fewer PEs than the link-oriented ones.

The bus-oriented architecture calls for a certain complexity in the structure of each node, which has to handle the interfacing on the common resource. As a result, the granularity of the PEs tends to be quite coarse. The few prototypes and systems built according to this paradigm consist of clusters of relatively powerful processors, based on commercially available, single-chip microprocessors. VLSI integration does not influence so much the interconnection structure of the system; rather, it establishes the processing capabilities of the node in terms of on-chip cache, memory management, floating-point unit, bus interfacing, and so on.

The perspective changes when one considers the link-oriented architectures. Here, the ultimate goal is true massive parallelism (thousands of PEs). The thrust here is therefore toward simpler PEs to allow for the integration of more nodes than ever into a single chip. The VLSI problem is here the layout feasibility of the various topologies and chip pinout.

There exists a well-developed theory of the computational complexity of the layout problem in VLSI;[23] it rests upon theoretical models, such as Thompson's,[24] whereby a chip is divided into a grid of vertical and horizontal tracks, separated by unit intervals and each processor is assumed to occupy a unit area, with wires routed through the tracks, but not over any processors unless connected to them. The two main problems that have been addressed are the area complexity, as a function of the number of processors, and the minimization of the longest edge length.

Among the hierarchical topologies here discussed, the one which has been studied to a certain extent is regular trees.

It is well known that the binary tree can be laid out in an area $O(N)$.[24] The first layout was the H-tree; however, it has been proved to have poor area utilization (50% in the limiting case), and it uses some nodes as crossing elements, a definite drawback when advanced integration techniques, such as Wafer Scale Integration (WSI), are used. More recently, other schemes have been introduced[25] that exploit the reconfiguration capabilities of the plain mesh of processors and achieve higher area utilization (90%), still using some nodes as switching elements (in the "tiles" approach), or even higher area efficiency (100%), abandoning the embedding approach and using recursive construction algorithms based on four basic elementary blocks which lay out a four-level binary tree.[26]

Other tree topologies, having branching factor $m=k^2$, have been studied in Ref. 27; they exhibit the same $O(N)$ area complexity of the binary tree. Moreover, these arrangements of k^2-way trees can also be profitably used for the quad-pyramid ($k=2$) structure, since the additional interconnections necessary for interlevel meshes can be routed on a different layer. As to the remaining structures in the tree family, it has been already observed that ringed trees, hypertrees, multitrees, and flip trees are actually instances of graphs; for this topology there exist theoretical results[23] on upper bounds for the area complexity, but little work has been done to produce actual layouts.

3.7.2. Applications

The bus-based families, for which there exist prototype and commercial systems (see Section 3.7.3), are characterized by a certain granularity in the architecture of the PEs. They have been and still remain a very good test-bed to develop operative systems to support MIMD processing. From the application point of view, they are amenable to very general computational paradigms and have been used successfully in vision (intermediate- and high-level tasks, such as handwritten script recognition and shape analysis).

The number of algorithms designed for trees, especially binary trees, is enormous. The spectrum ranges from parallel prefix computations, such as associative binary operations ($+$, x, min, max, etc.), to grammar parsers,[28] to data base and dictionary operations.[29] Quad-tree algorithms[30] also map very well on regular k^2-trees machines. Whenever the tree offers additional links, such as in the hypertree, other kinds of computations become feasible, because of the symmetry of the n-cube interconnections, fast Fourier transform (FFT)

and sorting, which can be formulated to rely on in-place binary operations easily carried out on neighboring PEs.[4]

The rich topology of hypernets can be used in many application domains.[7] Among the *data-parallel* algorithms, in which a subnet is assigned a certain subproblem of reduced size and all subnets cooperate to produce the solution, are *recursive doubling techniques* for binary commutative and associative operations, where an (n,d) structure is used primarily to emulate an n-level binary tree; *ascend–descend techniques* for bitonic merging, sorting, convolution, matrix multiplication, and FFT, for which the hypercube-type connections are better utilized in the hypernet structure than in the hypercube itself. In the domain of artificial intelligence, hypernets have been proposed to implement memory-based reasoning according to the similarity induction paradigm, as opposed to production rules-based reasoning and also to embed semantic networks, which require a *data-dependent* mapping.

3.7.3. Prototypes and Machines

The family of bus-structured systems here described is not represented exactly in any prototype or proposed system. However, the arrangement of a number of PEs in clusters around a common bus and the hierarchical interconnections of such clusters has been followed in at least two systems, the Cm*[31] and the *elaboratore multi mini associativo* (EMMA) family,[32] the latter also a successful commercial product.

In the broad family of link-oriented architectures, the structures upon which actual systems have been based are the regular tree and the bin and quad pyramids. The former has been used more as a component of a larger and possibly heterogeneous system, as in Non-Von[33, 34] and NETRA,[35] than as a standing, homogeneous machine, as in DADO.[36]

In the Non-Von system, a complete binary tree of simple processing elements (SPE) is connected to a set of intelligent disk processors at a particular, fixed level of the tree; moreover, each PE has an additional I/O port through which the tree can be reconfigured into other topologies. The system is designed to support three types of interprocessor communications: global bus communication, whereby the controller broadcasts data to all SPEs or fetches one datum from a single SPE; tree communication, which takes place among physically adjacent processors; and linear communication, in which the whole tree is reconfigured as a linear array of SPEs. The latest version of the prototype system, Non-Von 3, was planned to have a single, 8-bit processor per chip.

In NETRA, the hierarchical interconnection network is used to create an

n-tree of distributing and scheduling processors (DSP) which controls the set of PEs organized into clusters of 16 to 64 processors each. These processors are linked to a global memory by an interconnection network. Most notably, the tree structure is used as the control substructure of the whole system, not as the computing engine part of it. Leaf nodes of the tree are the finest granularity structure of control; the overall system therefore supports both a single-instruction, multiple-data (SIMD) and a MIMD processing capability.

The Columbia University DADO system has been designed as a pure binary tree machine, targeted at handling expert production systems. A board will contain a six-level tree, where each node is planned to have substantially more processing power than that of Non-Von.

Among the pyramid family, a few proposals have reached the prototype stage. With a single exception, all systems adopt the quad-pyramid structure. They differ in the level of integration and in the interconnection scheme used at board level more than in the processing capabilities of the PEs. These systems will be described in detail in Chapter 5.

3.8. CONCLUSIONS

In order to exploit hierarchical processes that can be useful in many application fields, several systems belonging to network computer, multicomputer, and multiprocessor families have been proposed and sometimes built following a hierarchical paradigm. In this chapter, the most popular topologies of these systems have been described and the main features and characteristics of these structures have been analyzed and compared on the basis of general loading assumptions.

REFERENCES

1. L. N. Bhuyan, Q. Yang, and D. P. Agrawal, Performance of multiprocessor interconnection network, *Computer* **22** (2), 25–36 (1989).
2. D. A. Reed and H. D. Schwetman, Cost-performance bounds for multimicrocomputer networks, *IEEE Trans. Comput.* **C-32** (1), 83–95 (1983).
3. R. A. Finkel and M. H. Solomon, Processor interconnection strategies, *IEEE Trans. Comput.* **C-29** (5), 360–371 (1980).
4. J. R. Goodman and S. H. Sequin, Hypertree: a multiprocessor interconnection topology, *IEEE Trans. Comput.* **C-30** (12), 923–933 (1981).

5 B W Arden and H Lee, A regular network for multicomputer systems, *IEEE Trans Comput* **C-31** (1), 60–69 (1982)

6 F J Meyer and D K Pradhan, Flip-trees fault tolerant graphs with wide containers, *IEEE Trans Comput* **C-37** (4), 472–478 (1988)

7 K Hwang and J Ghosh, Hypernet a communication-efficient architecture for constructing massively parallel computers, *IEEE Trans Comput* **C-36** (12), 1450–1466 (1987)

8 F Devos, A Merigot, and B Zadovique, Integration d'un processeur cellulaire pour une architecture pyramidale de traitement d'image, *Rev Phys Appl* **20,** 23–27 (1985)

9 V Cantoni, M Ferretti, S Levialdi, and R Stefanelli, PAPIA pyramidal architecture for parallel image analysis, Proc 7th Symp Computer Arithmetic, Urbana Il, 1985, pp 237–242

10 V P Nelson, Fault-tolerant computing fundamental concepts, *Computer* **23** (7), 19–25 (1990)

11 C R Lang, The extension of object-oriented languages to a homogeneous concurrent architecture, TR 5014, Computer Science Dept , California Institute of Technology, Pasadena (1982)

12 D P Bhandarkar, Analysis of memory interference in multiprocessors, *IEEE Trans Comput* **C-24,** 897–908 (1975)

13 J P Hayes, A graph model for fault tolerant computing systems, *IEEE Trans Comput* **C-25** (9), 875–883 (1976)

14 C L Kwan and S Toida, An optimal fault tolerant realization of symmetrical hierarchical tree systems, *Networks* **12,** 231–239 (1982)

15 C S Raghavendra, A Avizienis, and M Ercegovac, Fault tolerance in binary tree architectures, *IEEE Trans Comput* **C-33** (6), 568–572 (1984)

16 M B Lowrie and W K Fuchs, Reconfigurable tree architectures using sub-tree oriented fault tolerance, *IEEE Trans Comput* **C-36** (10), 1172–1182 (1987)

17 C H Sequin, Single chip computers, the new VLSI building blocks, Proc VLSI Conf , California Institute of Technology, Pasadena (1979)

18 D P Agrawal and V K Janakiram, Evaluating the performance of multicomputer configurations, *Computer* **19** (5), 23–37 (1986)

19 V Cantoni and S Levialdi (eds), *Pyramidal Systems for Computer Vision,* Springer-Verlag, Berlin (1986)

20 A Rosenfeld (ed), *Multiresolution Image Processing,* Springer-Verlag, Berlin (1984)

21 J R Amstrong and F G Gray, Fault diagnosis in a boolean *n*-cube array of microprocessors, *IEEE Trans Comput* **C-30** (8), 587–590 (1981)

22 V Cantoni, M Ferretti, and L Lombardi, A comparison of homogeneous hierarchical interconnection structures, Proc IEEE **79** (4), 416–428 (1991)

23 J D Ullman, *Computational Aspects of VLSI,* Computer Science Press, Rockville, MD (1984)

24 C D Thompson, A complexity theory for VLSI, Ph D dissertation, Carnegie-Mellon Univ , Dept Computer Science (1980)

25 D Gordon, Efficient embedding of binary trees in VLSI arrays, *IEEE Trans Comput* **C-36** (9), 1009–1018 (1987)

26 H Y Youn and A D Singh, On implementing large binary tree architectures in VLSI and WSI, *IEEE Trans Comput* **38** (4), 526–537 (1989)

27 N Ahuja, Efficient planar embedding of trees for VLSI layou, Proc 7th Int Conf Pattern Recognition, Montreal, 1984, pp 460–464

28 D L Milgram and A Rosenfeld, Array automata and array grammars, Proc IFIP Congress 1971, pp 166–173, North-Holland, Amsterdam (1971)

29. T. A. Ottman, A. L. Rosenberg, and L. J. Stockmeyer, A dictionary machine (for VLSI), *IEEE Trans. Comput.* **C-31** (9), 892–898 (1982).
30. H. Samet, The quadtree and related hierarchical data structures, *Comput. Surv.* **16,** 187–260 (1984).
31. R. J. Swan, S. H. Fuller, and D. P. Siewiorek, Cm*—a modular multiprocessor, Proc. AFIPS Nat. Comp. Conf., 1977, pp. 637–663.
32. E. Appiani, G. Barbagelata, F. Cavagnero, B. Conterno, and R. Manara, EMMA2, an industry developed hierarchical multiprocessor for very high performance signal processing applications, Proc. 1st Int. Conf. Supercomputing Systems, St. Petersburg, FL, 1985.
33. D. W. Shaw, Organization and operation of a massively parallel machine, in *Computers and Technology* (G. Rabat, ed.), North-Holland, Amsterdam (1986).
34. H. A. H. Ibrahim, J. R. Kender, and D. E. Shaw, On the application of massively parallel SIMD tree machines to certain intermediate-level vision tasks, *Comput. Vision, Graphics Image Process.* **36,** 53–75 (1986).
35. M. Sharma, N. Ahuja, and J. H. Patel, An architecture for a large scale multiprocessor vision system, in *Parallel Computer Vision* (L. Uhr, ed.), pp. 87–105, Academic Press New York (1987).
36. S. J. Stolfo and D. P. Miranker, The DADO production system machine, *J. Parallel Distributed Comput.* **3** (2), 269–296 (1986).

Chapter 4

A Taxonomy of Hierarchical Machines for Computer Vision

This chapter expands the notion of hierarchy by analyzing a variety of existing and proposed hierarchical systems which at various stages match the computational structure of a general computer vision task. Such systems have been based on different paradigms (Pipeline, SIMD, Multi-SIMD, MIMD, etc.), often mixed in various ways. A taxonomy will be presented in order to introduce a number of different families of these machines. The taxonomy is based on two hierarchical levels: the first splits the systems into homogeneous or heterogeneous ones according to the capability of processing modules; the second is based on the ways of coupling the modules and on the interconnection networks (tight–loose, compact–distributed, fixed–reconfigurable).

4.1. PARADIGM FOR COMPUTER VISION

The huge amount of visual sensory data produced even by standard acquisition systems cannot be processed in near real time by means of the sequential von Neumann machine and requires massively parallel systems. Moreover, the computational paradigm in the image analysis process varies widely; it is therefore difficult to find a unique architecture which is able to cope with the different requirements.[1] More specifically, the data structure and the computational structure change as shown in Figure 4.1, which highlights the hierarchical nature of the three processing stages briefly described subsequently.

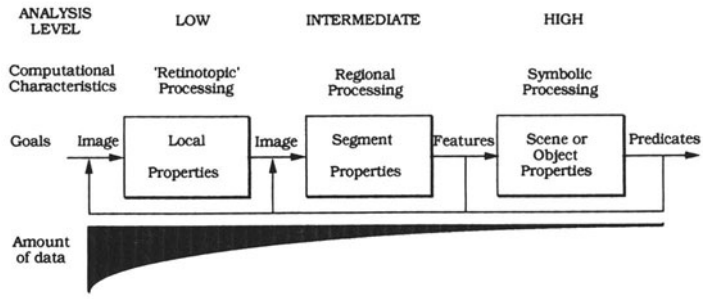


Figure 4.1. Three computational steps transform image pixels into predicates describing a scene: (i) a preprocessing stage; (ii) an intermediate description; (iii) the final interpretation.

4.1.1. Preprocessing

The goal of preprocessing is to tune up the data for successive analysis. This typically amounts to enhancing the image, by sharpening the contours, suppressing the background, removing noise, etc., or involves restoring it, by removing geometrical distortions, by amending optical defects, etc. The operation involved corresponds to an "image-to-image" transformation, so the data structure is fixed: the original bidimensional array of pixels. From the computational point of view, three different operation types can be considered: point-dependent operations (thresholding, requantization, encoding, etc.), local operations (filtering, template matching, convolution, geometrical correction, relaxation, etc.), regional and global operations (digital transforms like Fourier, Hadamard, Haar, etc.).

Most of the computations quoted share the following characteristics:

- *Spatial invariancy* The same sequence of instructions is executed on all points of the image, i.e., it is data independent.
- *Locality* At each point the final result is dependent only on a limited window around it.
- *Fixed data structure* The result still remains an image array.

Because of its affinity with the peripheral stages of the human vision system, this phase is usually referred to as *low-level processing* and some authors call it *retinotopic* processing.

4.1.2. Intermediate-Level Processing

The final goal of intermediate-level processing is the extraction of salient features and pertinent primitives of the image components.[2] In order to achieve

this the first processing phase is the *segmentation* of the image, i.e., the identification of homogeneous parts. Different criteria can be used to establish its homogeneity, the more common ones being based on morphological or chromatic characteristics, position in space, movement, etc.

The second phase is the description of segments. For each segment, some features are extracted on the basis of the gray-level statistics (histogramming, cooccurrences, etc.) and other parametrization techniques (moments, shape descriptors, etc.) by means of basic operations such as medial axis transformation, skeletonization, and edge following.[3]

This intermediate level corresponds globally to an ''image-to-features'' transformation. Following this, the data structure, initially a bidimensional array matching the image, changes toward more flexible structures, such as arrays or lists of features, and adjacency graphs. The choice of the features and of the proper data structure, which sets the level of abstraction for the content description of the image, is the critical issue.

From a computational point of view, it is important to note that often the operations required at this step realize regional processes which are *a priori* not diameter limited (local operations typical of the low-level phase still remain, but the majority of the computations depend on global support). The processing control at the intermediate level shows a transition from data parallelism to task parallelism. This is a peculiarity which clearly distinguishes this intermediate stage and compromises the efficient tuning of architectures to the tasks in question.

4.1.3. High-Level Processing

The final goal of high-level processing is the identification of image contents. Two main phases can be distinguished: symbolic description and scene understanding.

In the former by working on salient features, a more compact description is developed which makes the relationships among the image primitives explicit. The data structures of this step are usually trees, graphs, semantic networks, frames, and so on, where items represent symbols at higher levels of abstraction with respect to the original feature.[4] This phase corresponds globally to a ''features-to-symbols'' transformation.

In the latter phase of scene understanding, the aim is to produce an interpretation of image contents on the basis of a symbolic description and of the knowledge about the scene context. The objects present in the scene are labeled and identified as are the relationships among them. In this phase the data structure pertains to knowledge representation. A correspondence must be found

between symbolic descriptions and an explicit or implicit model of the scene to be analyzed. From the computational point of view, high flexibility is required to structure, coordinate, and integrate the different matching processes that can be applied concurrently. The amount of data to be dealt with can be huge, not so much for the symbolic scene description but mainly because of the knowledge required for the context. ''Scene understanding'' corresponds overall to a ''symbols-to-predicates'' transformation.[5]

Often in this phase some complementary information from previous stages are needed to produce new measurements or parameter values, or even change parameters once a partial interpretation has occurred. That is, image understanding is not a straightforward linear process but relies on feedback and backtracking procedures.

The term *high-level vision,* which usually denotes this last stage of the paradigm of computer vision, hints at the processes carried out by the upper layers of the vision system in humans.[6]

4.1.4. Three Possible Computational Frameworks

The logical sequence of the three phases described introduces a hierarchy in which the level of abstraction grows while the amount of data decreases. Three computational paradigms have been proposed to accomplish the hierarchical steps mentioned:

- The first follows a *data-driven,* bottom-up approach, in which control flows from low-level to high-level processing (each step works independently and passes its output onto the next step).
- The second follows a top-down approach, in which processing is *goal driven* and the control flows from a high level to a low level.
- The third is a more general heterarchical approach, as mentioned in describing the last processing stage; it is shown in Figure 4.1. In this hybrid case, the previous two solutions are interleaved on the basis of partial results; inconsistencies, ambiguities, or incomplete recognitions are detected and solved by feedback to the previous steps.

4.2. TAXONOMY OF HIERARCHICAL MACHINES

As claimed by many authors, high speedups can be obtained by matching the architecture to the data structure and/or to the computational structure.[1, 7] However, in computer vision, the data structure changes from a ''large,'' fixed

array of pixels of the initial steps to "small," flexible structures of the high-level steps. The computational structure also varies; the first steps deal with image-to-image transformations, identical operation applied to all pixels, and fine granularity (the computation involved for one pixel requires the analysis of its own local context). The last steps deal with data-dependent (asynchronous) operations, sometimes intrinsically sequential and recursive, with coarse-granularity communications and flexible data structures.

The SIMD and pipeline architectures are, of course, well suited for the low-level steps that require data-independent, synchronous, fine-grained operations; however, there are also among these steps tasks for which a more flexible structure would be preferable (typical low-level data-dependent operations are restoration with spatially variant point-spread function, split-and-merge techniques, etc.).

The MIMD architecture is not suited to synchronous operations but to high-level processing steps, which are usually composed of irregular, asynchronous, and coarse-grained subtasks.

For these reasons several hierarchical solutions have been proposed based on Pipeline, SIMD, Multi-SIMD, and MIMD paradigms mixed in various hierarchical ways.[8] In Figure 4.2 a taxonomy of the hierarchical systems which have been so far proposed is presented. Even the taxonomy is based on a

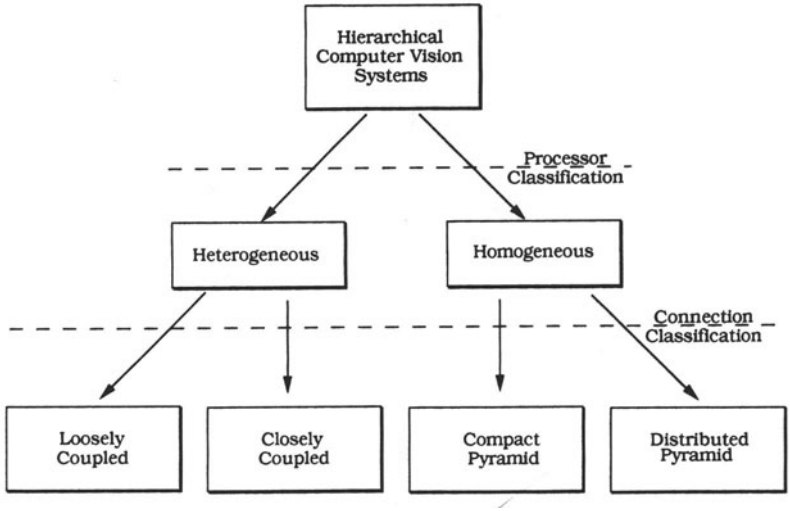


Figure 4.2. Classification scheme of the hierarchical systems. Two levels logically characterize the taxonomy: the first refers to homogeneity and the latter to type of coupling.

hierarchical scheme: the first criterion partitions the systems into homogeneous and heterogeneous classes; the second specializes the homogeneous family into compact and distributed architectures, and the heterogeneous case is split by coupling the components into loosely coupled and tightly coupled categories.

As shown in Chapter 1, in the theory of hierarchical modular systems, elements which go from the lowest to the highest levels of the hierarchy are given an increasing value. This fact, in many proposed hierarchical machines, leads to heterogeneous systems, in which more powerful processors are selected for the higher hierarchical levels. This specialization of the processors according to level contrasts with the system modularity: for this reason many homogeneous systems have been proposed as well.

In this last family of machines, basically two different kinds of approaches have been used: one involves architectures characterized by large-scale systems composed of very simple elementary processing elements (PEs), the so-called megamicrocomputers[9]; the other involves architectures characterized by very powerful processing units (PUs), used in fairly large amounts (i.e., hundreds, but still two orders of magnitudes less than in the previous case), which is enough to justify the term *massive parallel systems.*

This dichotomy is reflected in the technology and in the granularity of the system. VLSI capabilities are used in the former case to integrate the largest possible number of PEs in a single chip; meanwhile in the latter case they are used to augment the processing features of the PUs in terms of on-chip cache, memory management and floating-point units, bus interfacing, and richer connectivity.

Concerning granularity, PE-based machines are conceived with one processor per pixel; these PEs are usually specialized for near-neighbor operations which are a relevant percentage of all the processing that they carry out. For these reasons, systems based on PEs are classified as fine-grained machines. In the other case each PU takes care of a subimage stored in its own local memory and predominantly computes block properties autonomously, with less frequent inter-PU data exchanges, thus embodying coarse-grained computations.

In the heterogeneous family the loosely coupled machines are *network centered;* i.e., the interconnection structure is a central resource which is in charge of managing the flow of information and which heavily influences the system's performances. The second family is characterized by tight-coupled systems, where intercommunications are realized by sharing memory or first-in–first-out (FIFO) queues. These systems can be classified as *processor centered,* since it is the processor which directly handles the data exchanges.

Subsequently a few details are given for each of the four families resulting from the two-level taxonomy.

4.2.1. Heterogeneous Loosely Coupled Class

The machine in the heterogeneous loosely coupled class is composed of several parts: usually a SIMD array is demanded for low-level image processing, while different processors (usually MIMD units) are needed for the intermediate- and high-level phases. These subunits are physically different and coupled by buses, links, or multistage networks. This kind of interconnection does not make information exchanges between the subparts easier, but, due to their independence, autonomous design, implementation, and debugging are possible. The loose interconnection between the SIMD and MIMD parts allows us to achieve the best performances when the system executes the rather coarse-grained computations which are mapped on the subunits.

At least two systems can be classified in this family even if both do not follow completely the paradigm just described: the Non-Von system has been shortly described in Section 3.7.3, and the PASM system, briefly introduced next, and described in more detail in Section 8.4.

The PASM machine, developed by Siegel[10] at Purdue University, consists of 1024 processing units organized into 16 groups; each group has its own control unit as shown in Figure 4.3. The apex of this three-level hierarchy is the system control unit. Adjacent groups may be dynamically configured so that they behave as a single-SIMD system. Presumably, for low-level image processing tasks, the system would be configured as a single-SIMD machine with 1024 PUs. Data exchanges in this machine are easy, but some overhead, with respect to a standard SIMD machine (e.g., if this is operating in the broadcasting–gating mode), may arise in low-level processing when all the SIMD subsystems are working in a tightly ''orchestrated'' mode, and neighboring access is needed. The interconnection network among the groups of leaves is

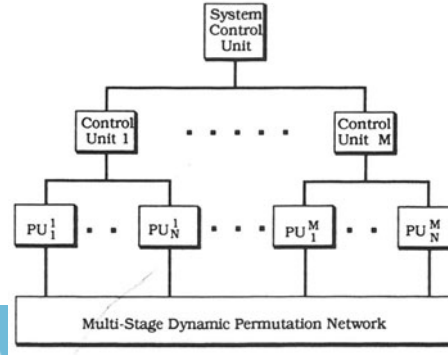


Figure 4.3. Simplified structure of the Partitionable SIMD/MIMD machine. PU clusters are driven hierarchically by a two-level controlling scheme. The first prototype has $M = 16$ and $N = 64$.

of the dynamic permutation type; this horizontal interconnection completes the vertical tree structure which implements the hierarchy.

4.2.2. Heterogeneous Closely Coupled Class

In the heterogeneous closely coupled family, the coupling among the system subunits is physically realized through shared memory and logically implemented by certain synchronization primitives. Also in heterogeneous closely coupled systems, two main units can be functionally identified for the low-level and higher-level tasks. These are a SIMD and a MIMD part respectively. Usually, a SIMD subsystem is devoted to each PU of the MIMD structure so that the two parts are integrated (e.g., Uhr's Array/Net[11] IUA[12], and WPM[13]).

In Figure 4.4 a paradigm for this family of machines is shown. The two subsystems are physically distinct and linked through as many dual-ported memories as there are processors within the MIMD part. Indeed several buses are necessary between the SIMD structure which exchanges its data with the memory in a synchronous mode (each bus is supplying information from/to a subset of SIMD PEs) and the MIMD structure which works asynchronously with the same memory. A large number of communication buses increases the capacity to exchange data while introducing a certain amount of interface complexity. In this machine each PU of the MIMD part is linked to four close neighbors in a mesh topology. Since each processor in the MIMD component interacts with several PEs in the SIMD part, the data parallelism in the two subunits is different: hence an interface with both multiplexing and buffer capabilities is needed. These buffers (the coupling memories in Figure 4.4) can be considered as external memories for both the MIMD and SIMD subunits.

4.2.3. Homogeneous Compact Pyramid

The homogeneous pyramidal structure has been formally defined, from the topological viewpoint, in Section 3.5.4. In this section the discussion will be restricted to the compact case. The architecture of this class of machines stems from two distinct but converging efforts: exploiting VLSI technology to produce very large arrays of PEs and taking advantage of multiresolution techniques which have proved to be extremely useful in image processing. Cellular arrays, as, for example, CLIP4,[14] MPP,[15] DAP,[16] and more recently massively parallel fine-grained systems, as the Connection Machine[17] and MasPar,[18, 19] build a network of fairly simple, bit-serial PEs (with the exception of MasPar, which has a nibble-based data path) by integrating a set of such processors in a single chip. The homogeneity of the processors is the key item for a success-

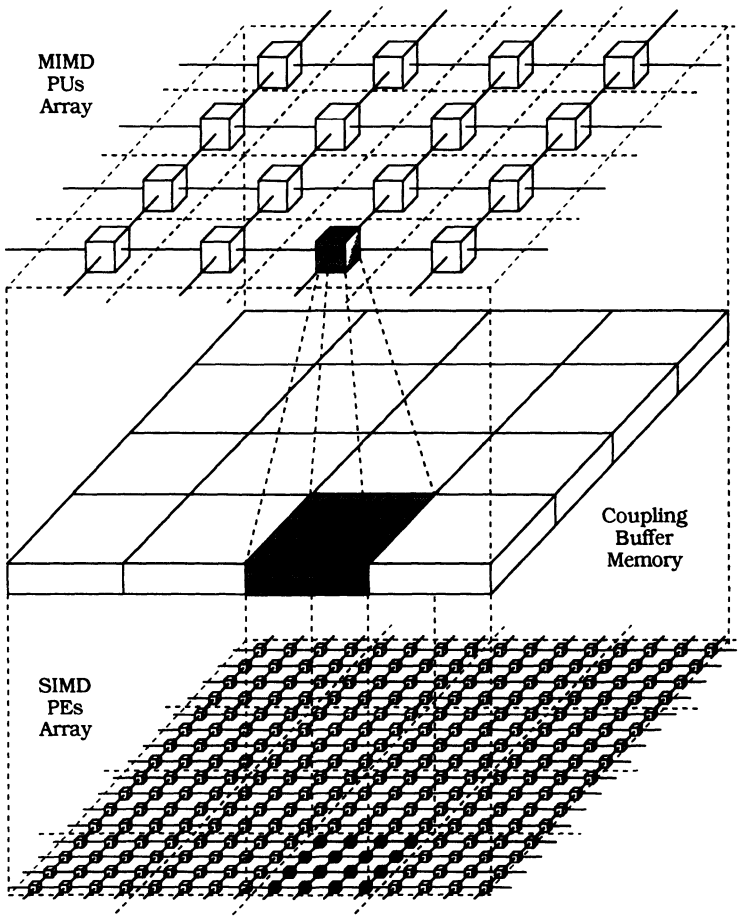


Figure 4.4. Simplified structure of a heterogeneous machine consisting of a MIMD and a SIMD subunit coupled by means of dual-ported shared memory.

ful VLSI realization. The current level of technology allows the construction of chips containing a number of PEs ranging in the hundreds. Obviously, the small silicon area allocated to each PE limits its processing autonomy: the array activity is completely "orchestrated" by a powerful external controller which broadcasts the instruction stream to the array. Furthermore, the interconnecting structure is fixed, and each PE communicates directly with its immediate neighbors in the underlying topology, which is a grid in cellular arrays, a hypercube in the Connection Machine, and a shuffle-exchange network in the MasPar (in addition to the grid network).

Nevertheless, a mismatch still exists between the largest array of PEs which is economically feasible and the size of the images to be processed. So, some problems arise because only a small portion of the image can be processed simultaneously: the approach is to distribute the image to the PEs one subimage at a time. Difficulties arise in the block border processors when performing local operations; in fact, the neighbor data of these PEs are not directly available. Moreover, if such operations are iterated many times, the set of active PEs shrinks at every step.

Homogeneous compact pyramids (see Figure 3.14) extend the cellular array topology by stacking arrays of decreasing size on top of each other.[20] As a consequence, this kind of architecture not only allows SIMD operations but also Multi-SIMD processing as well, provided that the controller of the system is capable of synchronizing data exchange among the adjacent layers. Among the machines in this family are (in chronological order) PCLIP,[21] PAPIA,[22] GAM,[23] SPHINX.[24]

4.2.4 Homogeneous Distributed Pyramid

An alternative way of conceiving a pyramidal computer is to use a limited number of identical but powerful PUs (usually in the order of tens). The capabilities of the PUs allow us to realize the topology in different ways: either a *network computer* solution[25] or a distributed shared-memory solution such as the Erlangen general-purpose array (EGPA) developed at the Erlangen University.[26]

A representative system for the general homogeneous distributed pyramid family is the quad pyramid of the EGPA, shown in Figure 4.5. This system is a three-level pyramid containing 21 processor memory modules (PMMs) with mutual memory access among neighbors. There are 16 PMMs in the lower level, 4 in the middle, and one supervisor PMM at the apex. Each PMM also has access to the memory of the four children. In this way, by means of common control blocks and mailbox techniques, interprocessor communications between neighboring workers and between supervisor and generic worker (in unidirectional mode) are achieved.

Due to the moderate number of PUs, when these machines are used for low-level image processing, a suitable approach is the distribution of the PUs throughout the image. Following this approach, each PU deals with a subimage, and, therefore, an overhead caused by the border problem and by the coordination of the PUs must be taken into account. With respect to the homogeneous compact pyramid case, their increased flexibility favors high-level processing at the expense of low-level processing. Access to the near neighbors of

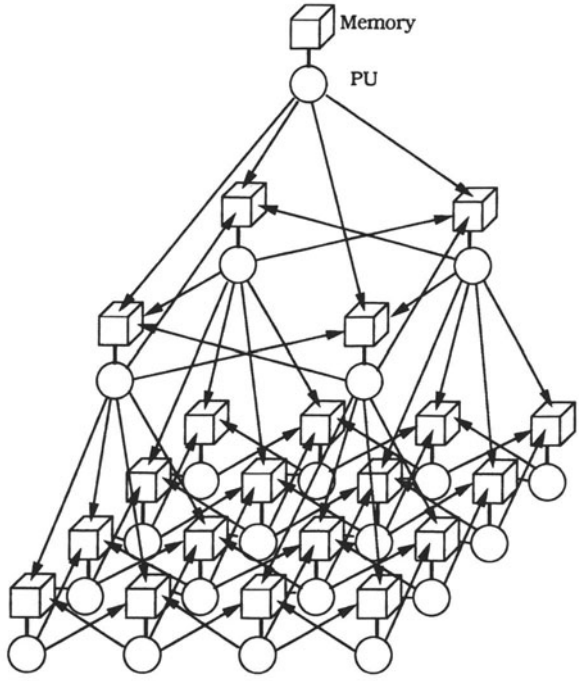


Figure 4.5. The distributed EGPA pyramid: each PU has access to its own local memory and to brother and children memories.

each pixel becomes serial, and, for many low-level tasks in which local operators are very frequent, this becomes a serious bottleneck.[27]

The high flexibility of these coarse-grained systems allows different operative modalities: the preeminent one is the MIMD mode, but in many practical applications data partitioning is implemented and the PUs execute multiple instances of the same instruction sequence in a Single-Program-Multiple-Data (SPMD) modality (···). Moreover, the PU interconnection can be conceived by a multistage dynamically reconfigurable network, which eases the mapping of the task onto available PUs.

4.3. CONCLUSIONS

In this chapter a few paradigms for creating a hierarchical system for computer vision have been introduced and organized in a two-level hierarchical

taxonomy. The discussion has only highlighted the main characteristics of the systems; later chapters cover in more detail some of the families which have been introduced here, by analyzing implementations issues, functional properties, and typical applications within the computer vision domain.

Presently, the systems that in one way or another fall within the families outlined in the taxonomy and that have at least reached the prototype stage do not always share all the distinctive characteristics which have just been outlined. In some cases, the architecture embodies the paradigm of its family quite extensively, while in others the match is only partial. There are even situations where an architecture, which is definitely outside the taxonomy, has been used to emulate an instance of a machine among those considered here. This case is also covered in a following chapter, since the hierarchical processing mode is considered effective even when the architecture at hand has no special feature to support it.

REFERENCES

1. V. Cantoni and S. Levialdi, Matching the task to an image processing architecture, *Comput. Vision, Graphics Image Process.* **22** (2), 301–309 (1983).
2. M. J. B. Duff (ed.), *Intermediate-Level Image Processing,* Academic Press, London (1986).
3. A. Rosenfeld and A. C. Kak, *Digital Picture Processing,* Academic Press, New York (1982).
4. D. H. Ballard and C. M. Brown, *Computer Vision,* Prentice-Hall, Englewood Cliffs, NJ (1982).
5. A. P. Pentland (ed.), *From Pixels to Predicates,* Ablex, Norwood, NJ (1986).
6. M. A. Fischler and O. Firschein, *Intelligence: The Eye, the Brain, and the Computer,* Addison-Wesley, Reading, MA (1987).
7. M. J. B. Duff (ed.), *Computing Structures for Image Processing,* pp. 1–14, Academic Press, London (1983).
8. V. Cantoni and S. Levialdi (eds.), *Pyramidal Systems for Computer Vision,* NATO ASI Series F, Vol. 25, Springer-Verlag, Berlin (1986).
9. L. D. Wittie, Efficient message routing in mega-micro computer networks, Proc. Third Symp. Computer Architectures, 1976, pp. 136–140.
10. H. J. Siegel, PASM: A reconfigurable multimicrocomputer system for image processing, in *Languages and Architectures for Image Processing* (M. J. B. Duff and S. Levialdi, eds.), pp. 257–266, Academic Press, London (1981).
11. L. Uhr, J. Lackey, and L. Thompson, A 2-layered SIMD/MIMD parallel pyramidal 'array/net,' Proc. Workshop on Computer Architectures for Pattern Analysis and Image Data Base Management, 1981, pp. 209–216.
12. G. R. Nudd, D. J. Kerbyson, T. J. Atherton, N. D. Francis, R. A. Packwood, and G. J. B. Vardin, A massively parallel heterogeneous VLSI architecture for MSIMD processing, in *Algorithms and Parallel VLSI Architectures* (F. Deprettere and A. Van der Veen, eds.), pp. 463–472, Elvesier, Amsterdam (1991).
13. C. C. Weems, E. M. Riseman, and A. R. Hanson, Image understanding architecture: exploiting potential parallelism in machine vision, *Computer* **25** (2), 65–68 (1992).

14. M. J. B. Duff, Review of the CLIP image processing system, AFIPS Conf. Proc., Vol. 47, NCC, 1978, pp. 1055–1060.
15. K. E. Batcher, Design of a massively parallel processor, *IEEE Trans. Comput.*, **C-29,** (9), 836–840 (1980).
16. D. J. Hunt, The ICL DAP and its application to image processing, in *Languages and Architectures for Image Processing* (M. B. J. Duff and S. Levialdi, eds.), Academic Press, London (1981).
17. W. D. Hillis, *The Connection Machine,* MIT Press, Cambridge, MA (1982).
18. T. Blank, The MasPAr MP-1 Architecture, Proc. IEEE Compcon, 1990.
19. J. R. Nickolls, The design of the MasPar MP-1: a cost effective massively parallel computer, Proc. IEEE Compcon, 1990.
20. C. R. Dyer, A VLSI pyramid machine for hierarchical parallel image processing, Proc. PRIP, Dallas, TX, 1981, pp. 381–386.
21. S. L. Tanimoto, T. J. Ligocki, and R. Ling, A prototype pyramid machine for hierarchical cellular logic, in *Parallel Computer Vision* (L. Uhr, ed.), pp. 43–83, Academic Press, Orlando (1987).
22. V. Cantoni, M. Ferretti, and S. Levialdi, PAPIA: pyramidal architecture for parallel image analysis, Proc. 7th Symp. Computer Arithmetic, Urbana, IL, 1985, pp. 237–242.
23. D. H. Schaefer, G. C. Wilcox, and V. J. Harris, A pyramid of MPP processing elements—experiences and plans, Proc. 18th Ann. Hawaii Int. Conf. System Science, Vol. 1, 1985, pp. 178–184.
24. F. Devos, A. Merigot, and B. Zavidovique, Integration d'un processeur cellulaire pour une architecture pyramidale de traitement d'image, *Rev. Phys. Appl.* **20,** 23–27 (1985).
25. L. D. Wittie, Communications structures for large multi-micro-computer systems, *IEEE Trans. Comput.* **C-30** (4), 264–273 (1981).
26. G. Fritsch, General purpose pyramidal architectures, in *Pyramidal Systems for Computer Vision* (V. Cantoni and S. Levialdi, eds.), pp. 42–58, Springer-Verlag, Berlin (1986).
27. P. E. Danielsson, Vices and virtues of image parallel machines, in *Digital Image Analysis* (S. Levialdi, ed.), pp. 47–59, Pitman, London (1984).

Chapter 5

Compact and Distributed Pyramids

Here we consider the pyramid hierarchical homogeneous architectures. When designing a pyramid structure, we may follow two different approaches: the first with fine granularity, where one processor per image pixel is conceived, and the second with coarse granularity, where one microprocessor is associated with an image block. Subsequently, a detailed description of the projects which reached the prototype stage will be given, while looking comparatively at the various salient features.

5.1. INTRODUCTION

In pyramid structures several editions of the same image at different resolution levels are available, thus supplying the capability to implement multiresolution approaches. In particular, planning strategies and multigrid methods can be effectively implemented.

As previously described, in the former case a solution of the problem is pursued at the higher levels after which the solution is successively refined step by step toward the base. If the second phase of refinement properly exploits the hardware structure, and the amount of computations required at the low resolution levels is restricted because of the small quantity of data, the speedup factor can be even greater than the number of processors (in this way the linear scaling assumption can be overcome).

The latter case is an outstanding peculiar implementation of the previous

approach. For example, consider the problem of solving partial derivative equations (described extensively in Section 10.4.1). After the construction of multigrid editions of the equations by reprocessing on the higher levels, a solution is found on one of the higher layers of the pyramid. This can be performed with few iterations because of the small amount of data. Then, by interpolation, the initial values for a refined solution on the next lower level are provided. Each partial result is usually achieved by means of a few recursive steps in each new layer until the final solution is obtained on the pyramid base.

In order to effectively implement these processes several pyramid machines have been proposed and built following the compact and distributed pyramid paradigms described in Sections 4.2.3 and 4.2.4 respectively.

The first group corresponds to SIMD machines in which several layers of identical processors are organized in a tightly orchestrated mode. The approach is to set up systems using the largest number of processing elements (compatible with economical and technological reasons) by executing the same operation in parallel, despite the simple-processing-unit. For this reason the word length is set at 1 bit and serial arithmetic is adopted. In the flat-array case this solution has been called image-to-processor distribution.[1] These systems are suitable for the implementation of planning strategies, multigrid solutions, and low-level image processing (even if border problems arise when the image size is greater than the matrix of processors in the base). However the implementation of high-level image processing is cumbersome and less effective.

The distributed pyramid uses an approach which is equivalent to the one called ''processor-to-image distribution'' in the flat-array cases and which leads to a ''small'' number of identical, powerful processors. In this case each processor for low-level vision tasks deals with a portion of the image (obviously also in these cases an overhead due to the border problem occurs). With respect to the previous solution, there are some general gains in flexibility and capability for high-level image processing in spite of the low-level processing power (there is a lack of efficiency in implementing propagation and all common recursive operations).

In what follows, the main features and characteristics of the most popular machines for the compact and distributed cases respectively will be introduced in more detail in Sections 5.2 and 5.3.

5.2. COMPACT PYRAMIDS

Alternative solutions are possible within this paradigm. These shall be described in different chapters: *physical pyramids* (this chapter), which match

the definition of the quad or bin pyramid in hardware and are therefore 3-D systems; *simulated pyramids* or augmented mesharrays (Section 7.1), which either use the bare flat array as a simulating architecture or add a minimal extra connection capability to ease the burden of data collection between PEs at different levels; and *virtual pyramids* embedded in a flat mesh structure (Section 7.3).

At least four systems exist which can be defined as true physical pyramids: PAPIA 1,[2] designed by a consortium of Italian universities, PCLIP,[3] in some papers also called HCL, of Washington University, GAM[4] from George Mason University, and SPHINX[5] by Universitè Paris Sud. They provide a good framework to analyze the characteristics of compact pyramids; in this context, they will be reviewed and the different solutions they adopt for the construction of a working machine will be highlighted. First, the basic features of the four pyramids are introduced and compared, focusing on interconnection topology, processing element capabilities, chip complexity, and system–prototype assembly. Subsequent sections describe the systems in detail, pointing out the components which integrate the pyramid in a fully functioning machine.

5.2.1. Interconnection Topology

As mentioned in Section 3.5.4, tessellation topology, degree of reduction between levels, width of support of each new element above the base, and number of levels are the four parameters which define pyramids.

All the aforementioned systems adopt square mesh tessellation. This connects the PEs within a plane in a grid; PAPIA 1, GAM, and SPHINX use 4-connectivity, while PCLIP uses 8-connectivity. As for the other parameters, two solutions have been followed: PAPIA 1, GAM, and PCLIP adopt the topology of the *quad pyramid,* in which hierarchical connections reduce the number of processors at each new level by a factor of 4. SPHINX uses a reduction factor of 2, which results in the *bin pyramid.* The designers of this system motivate their choice, among other reasons, with the observation that the processing unit of the PE is usually a two-input device. The two topologies are show in Figure 3.14.

No effort has been made to build systems according to the definition of overlapped pyramids. The extra connections required to provide for overlapped, hierarchical domains would increase the complexity, already relevant, of such systems, and it can be argued that the benefits over simulated solutions might be minor, even for segmentation tasks.[6]

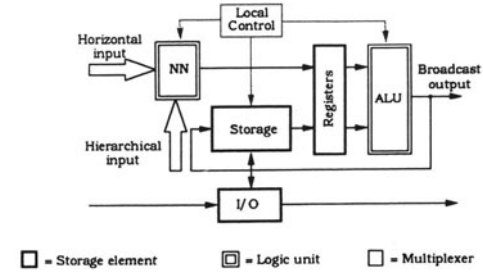


Figure 5.1. Block diagram of the PE of a compact pyramid system.

5.2.2. Processing Element Capabilities

Since all these systems are designed to take advantage of VLSI technology to integrate more PEs into a single chip, the architectures tend to be rather simple and so all adopt bit-serial processing. Nevertheless, it is worthwhile comparing the four systems from the point of view of PE architecture. To do this, one can identify five basic items: near-neighbor access, working storage organization, arithmetic and logical unit, local control, and image loading and unloading (see Figure 5.1).

5.2.2.1. Near-Neighbor Access

The near-neighbor-access function, which is the most unique capability of all fine-grained cellular architectures, is even more important in systems aimed

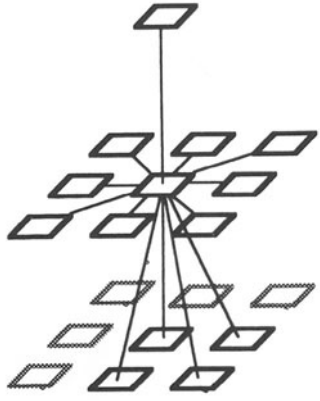


Figure 5.2. Near neighborhood of a PE in a compact quad pyramid with 8-connectivity, without overlapping. Among the lateral neighborhood there are three siblings, and the hierarchical neighbors are subdivided into parent and children.

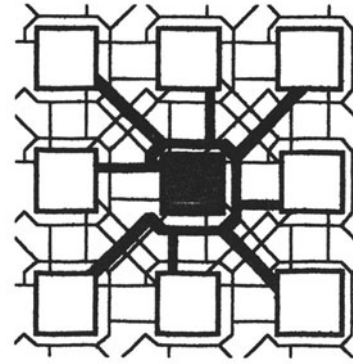


Figure 5.3. Partial block diagram of the interconnection involved with the broadcasting–gating near-neighbor operation. The scheme refers only to the wiring of the central PE and is limited to the lateral interconnection in 8-connectivity.

at multiresolution processing, where interlevel data exchanges can be the dominant activity. Given the total near neighborhood (NN) of a PE (see Figure 5.2) which consists of processors on the same plane (lateral neighbors or *brothers*), on the plane below *(children),* and on the plane above *(parent),* one can adopt two different strategies to access the information:

- *Multiplexing.* Data are fetched from neighboring PEs without processing. Data come directly from a neighbor to the data-in line of each PE. In this way, a piece of NN data at a time is available to the PE.
- *Gating.* A Boolean function is applied to data coming from NNs with a defined enabling vector. In this way an arbitrary subset of the NNs can be locally manipulated at once. Until now, no autonomy has been given to this operation: both the subset and the Boolean function are fixed for the full array (a proposal which introduces a first degree of autonomy is YUPPIE,[7] in which a binary interconnection status register selects between two possible elementary paths). In Figure 5.3 the basic scheme of the broadcasting–gating technique is shown. Here a detailed description of NN operations for this family of fine-grained machines follows.

a. Near-Neighbor Operations. A general expression of the code of these machines for NN operations is

$$Z = \Phi(X,T) \text{ edge } E \tag{5.1}$$

where Z is the PEs internal register which stores ALU results; $\Phi(\cdot,\cdot)$ represents one of the ALU Boolean operations (usually the set includes AND, OR, NOT,

EXOR, etc.), X is the set of internal registers which can feed the ALU, T is the result of the NN data combination; finally E indicates the value transmitted to the border processors from the external frame.

Note that, in particular, it is possible to extract data from just one neighbor by using a minterm in the NN enabling vector.

A class of operators that directly exploits the NN access capability (in just one clock cycle for structuring elements contained in the 3×3 subarray) is that belonging to basic mathematical morphology; namely erosion and dilation (the "hit or miss" operator requires a few clock cycles). Alternative implementations of the basic morphological operations have been used (e.g., in Yoda *et al.*[8] for erosion, dilation, and restricted broadening applied to automatic inspection problems) with higher computation time but with less expensive hardware. What makes the fine-grained solution more attractive is discussed in the next section.

b. Recursive NN Operations. In fine-grained machines PEs have direct access only to nearest neighbors, and data transfer between nonadjacent PEs requires a number of steps which are given by the number of PEs included in the connecting path between transmitter and receiver; in fact, exchanges take place only by adjacency (up to now only the CM[9] deviates from this). When the system uses the gating technique, data exchanges between distant PEs are implemented by iteratively applying the previous neighboring operation.

A basic feature available only with the gating technique is the capacity to apply an NN operation in recursive mode: the resulting Z register of Eq. (5.1) belongs to the set X of the input registers and is precisely that broadcasted to NNs. The operation is maintained until a stable state is reached. This feature is often called "propagation,"[10, 11] because one of the most important applications is to identify a complete segment propagating the signal from seed(s) within the connected component by adjacency.

This feature has been implemented in different ways; a common case is

$$Z^0_{t+1} = \Phi(X^0, \underset{i \epsilon \mathrm{NN}}{\mathcal{T}} [g^i \cap Z^i_t]) \tag{5.2}$$

where g is the gating vector, superscripts give the neighboring position (adopting the convention of Figure 5.4), subscripts express the recursion's step, and $\mathcal{T}$ represents the NN function.

The basic operation of propagation is given by

$$Z^0_{t+1} = X^0 \cap (\underset{i \epsilon \mathrm{NN}}{\cup} Z^i_t) \tag{5.3}$$

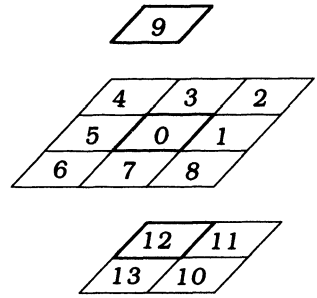


Figure 5.4. Complete hierarchical near-neighborhood labeling convention.

which is easily derived from Eq. (5.2). This operation allows us to consider the connected component as the atomic data for the investigations.

c. Implementations in Actual Systems. Of the four systems under analysis, PAPIA 1 and PCLIP adopt a complete gating approach; however, they exhibit some differences in that PAPIA 1 PE has special circuitry devoted to the implementation of the NN function, which is distinct from the processing unit, while HCL compresses these two units into a single facility called the *matching unit.* GAM, which is based on the chip designed for MPP, inherits the multiplexing approach of that system and extends it to handle vertical communications as well. SPHINX also uses the multiplexing solution for intralevel communication, while it adopts a mail box register approach for interlevel data movements. Details and comparisons of these different implementations will be discussed later.

In reference to the class of operations which involves multibit data transfer, multiplexed and gated NN access are perfectly equal, since no parallel access is possible in either case. On the whole, architectures provided with gated NN access exhibit a higher flexibility; however, in the domain of multiresolution processing, where algorithms exploit at least integer arithmetic, the advantage tends to be less evident.

5.2.2.2. Working Storage Organization

The storage resources accessible to the processing unit of the PE include registers, local memory, and external memory, if any.

While all the architectures analyzed here are provided with a few Boolean

registers to hold input and output operands, only two, PAPIA 1 and GAM, contain variable-length shift registers (2 and 1 respectively). They allow increased throughput in those operations, such as convolutions, in which partial results can be recirculated through the processing unit. The advantage over other architectures is relevant if the access time to local or external memory becomes the dominant factor in processing time.

The local or external memory available in these systems provides, per PE, 256 bits in SPHINX (local, and 8 kbits external), 1 kbit in PAPIA 1 (external), and 8 kbits in PCLIP and GAM (both external). To ease system design, memory is static. As for memory access, these architectures are based on one-address instruction formats, where the address construction is performed by the machine controller. The SPHINX system is an exception, since the PE executes three-address instructions; to prevent excessive number of I/O pins to transmit those addresses, the PE contains a set of autoincrementing–autodecrementing pointers.

5.2.2.3. Arithmetic and Logical Unit Structure

Boolean and arithmetic capabilities are features of the ALU units in all systems. They are designed to handle bit-serial processing and therefore are based on extremely simple circuits. A full adder is all that is required to perform two's complement serial arithmetic. Actually, PAPIA 1 and SPHINX are equipped with a unit which integrates both Boolean operations and arithmetic ones, while a minimal support is added for comparison and multiplication. GAM architecture uses, as mentioned above, the massively parallel processor (MPP) PE that separates the Boolean processor from the full adder. A unique design is that of PCLIP, whose processing capabilities are based on a single circuitry, the matching unit. It computes the logical AND (OR) between one of the local registers and the subset of the complete neighborhood of each PE that matches a ''pattern,'' specified in terms of 1 (the neighbor value is used), 0 (the neighbor value is discarded), or *D* (don't care condition; the neighbor value can be either 0 or 1).

In PAPIA 1 and GAM the availability of the shift registers, previously described as working storage, increases the processing capabilities and allows us to tune the structure of the PE to gray-level resolution in bits/per pixel of the image.

5.2.2.4. Local Control

Execution flow in fine-grained parallel systems is dictated by an external control unit, which holds the program to be executed and broadcasts it to all

the PEs. As elements of a SIMD system the PEs have limited local autonomy; Li[7, 12] identifies three types of autonomy, one for each of the three basic components of the PE structure shown in Figure 5.1.

- *Operation autonomy*. Truly autonomous PEs could execute different instructions, but they would also require local program storage and a local instruction decoding unit. This contradicts the trend of integrating an ever-larger number of PEs onto a single chip. The common trade-off is the inclusion of a special Boolean register, which holds an *activity bit* in the PE structure. According to the status of this register, PEs are subdivided into two sets: enabled PEs, which execute the current instruction, and disabled ones, which stay idle. Sometimes a somewhat more powerful model is proposed[13]: the activity bit conditions the execution of the currently broadcast instruction and causes one of two complementary operations to be carried out, such as additions versus subtractions, logical AND versus logical OR, etc.
- *Address autonomy*. Each PE accesses its own local memory using a private address. This is usually implemented with an offset mechanism, in which the broadcast address, common to all PEs, is modified by local data. Of the four systems discussed, the only one which exploits this feature is SPHINX.
- *Connection autonomy*. Each PE processes a different subset of its neighbors which is chosen among a few possible patterns according to the status of a local connectivity register. This feature is not implemented in any of the four pyramid systems mentioned. A limited implementation of this in a massively parallel, flat, fine-grained system can be found in YUPPIE,[7] where the connectivity register is binary and where the two possible patterns must satisfy certain constraints.

A last feature of the PE structure which contributes to the control strategy is the status register. This single-bit storage element is used to produce a global signal for the controller. Details on how to generate and use this global signal follow in Section 5.2.3. The data produced by each PE for such a function are static in three of the systems considered. Only PAPIA 1 PE is also capable of outputting a status change signal specifying whether the last instruction executed has modified the status register.

5.2.3 System Configuration

The systems analyzed here share a number of very peculiar features which distinguish them from general-purpose, coarse-grained systems. First, the hierarchical topology involves a complex management of multilevel operative

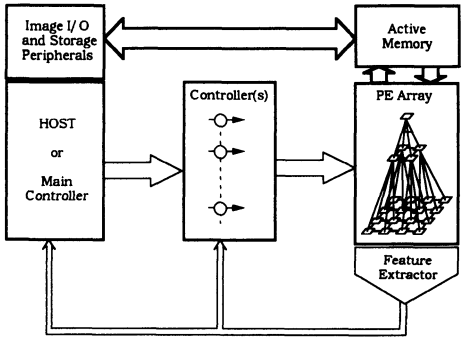


Figure 5.5. Block diagram of a general compact pyramid system. Only component units peculiar to this system are drawn; meanwhile auxiliary components such as standard image I/O subsystems are not included.

units; second, the serial structure of the PEs means that the raw elementary data is actually the bit plane, while the standard way of handling images is pixel oriented; third, all currently fabricated, or even designed, systems present a mismatch between the base array of PEs and the standard image size. Furthermore, any application requires both scalar and massively parallel operations, which obviously means using separate units. Finally, a global feedback to the controller is needed to operate on the image content at higher levels of abstraction. These peculiarities lead to a composite system structure; an overall block diagram of a general compact pyramid system is given in Figure 5.5.

5.2.3.1. The Controller

A key unit of the system is the controller, whose main functions are multitasking needed to manage the multilayer structure; distinguishing the parallel code from the scalar one, generating the pyramid code; handling of the global feedback coming from the pyramid array; and host interfacing or, for stand-alone machines, complete operating system support.

PAPIA 1 and SPHINX systems have operation autonomy at plane level; i.e., all layers can simultaneously execute different tasks, and the controller is in charge of the resulting Multi-SIMD modality. Details of the strategies used in these systems to manage tasks interaction are discussed in Chapter 9.

Code generation for the pyramid unit is a heavy burden for the controller who receives pixel-oriented macroinstructions from the host, handles the scalar code, and expands each parallel macroinstruction into the proper sequence of bit-oriented nanoinstructions to be broadcast to the pyramid PEs. A simple example is the sum of two images at a given resolution level: this single macroinstruction is expanded into a number of one-bit elementary additions as re-

quired by the bit length of the pixel. For these reasons a high-speed microprogrammed control unit is usually required.

An important feature of the control strategy in such systems is the handling of branching in the program flow managed by the controller. To obtain such a capability it is necessary for a single global status to be extracted from the parallel unit; most systems, particularly those studied here, are equipped with circuitry, named *global OR,* whose function is to build the logical summation of the contents of all the status registers of the PEs. This result can be used by the controller to make tests. PAPIA 1 can output the logical OR of a bit which is set in each PE if the last instruction has changed the status register: this allows for an easier implementation of the propagation function described in Section 5.2.2.1b). Besides these global tests, a special circuitry is introduced in some systems (e.g., GAM) to compute and deliver regional and global features to the controller, such as pixels counting, local arithmetics, etc.

In most cases the pyramid is considered as an attached processor and the controller is the natural host interface. A self-host configuration is seldom used to produce stand-alone systems; in these cases the controller takes on all the functions of the host, including peripheral and storage management.

5.2.3.2. The Active Memory

The image input–output management takes care of the following: mismatch between base array PE size and data size; transformation of the data from byte to bit-plane formats; image windowing and region of interest extraction; and effective bit planes loading and unloading.

Until today, the systems built or designed even in their largest configuration have a 128×128 base; this size, even if remarkable, is smaller by one order of magnitude than the usual image size. Image operations are implemented by sequences of block operations; it is up to the active memory unit to tackle this problem, which obviously reduces system performance. The practical solution to this problem is cumbersome, especially for local operations in which border effects ensue; these effects are not *a priori* limited in space if the local operations are executed recursively. Nevertheless, the mapping of the image on the base can be transparent to the users, thereby creating a virtual PE environment.

The basic function of the active memory is changing the format from the pixel environment of acquisition, monitoring, and storing subsystems to the bit-plane environment of the pyramid PEs. This trivial operation is crucial for system performance and alone justifies the active memory unit; sometimes this

operation has been implemented in *ad hoc* silicon chips (e.g., the corner turn buffer from the GAPP flat-array PEs).

In some implementations the active memory can read out the block to be fed to the pyramid base starting from an arbitrary location within the image (dynamic windowing). With the addition of a small amount of hardware the read-out address generation can be driven by a scanning step which is larger than unity; in this way, the input image can be arbitrarily subsampled. Combining the windowing effect and the subsampling features, the bit plane $B(i,j)$ transmitted to the pyramid base is

$$B(i,j) = I^r(x_0 + ki;\ y_0 + kj) \qquad \forall i,j \quad 1 \le i,j \le L \tag{5.4}$$

where I is the image stored in the active memory, r is the bit position, k is the sampling step, L is the linear size of the pyramid base, and (x_0,y_0) are the coordinates of the reference corner of the chosen window.

Moreover, in each PE, at least three of these systems foresee the presence of a special Boolean register for bit-plane image I/O. They are arranged in the form of unidirectional, distributed shift registers along one of the dimensions of the base array of the pyramid. In this way loading and unloading of images is performed with column (or row) parallelism and can, at least in principle, be concurrent with other processing activities in the pyramid. Nevertheless, the image input–output still remains a bottleneck for compact systems.[14]

5.2.3.3. The Host Computer

Very specialized parallel processors, such as the pyramid machines considered here, cannot efficiently support a complete, flexible, and efficient user programming environment. The controller, which is burdened with the heavy chores of feeding the pyramid nanoinstruction stream, managing interlevel data synchronization, handling the global array feature, and other scalar duties, usually does not have enough resources to serve as a developing machine. Therefore, it is the host computer, usually a conventional serial processor, that supports applications development: specially tuned compilers for high-level languages, debugging tools, and visualization and simulation environments. These are the software platforms that transform the high-speed parallel pyramid unit into an effective, real-life, application-solving machine.

In addition to software development, the host interfaces the active memory and the controller with standard peripherals (high-capacity and high-speed disks, TV cameras, monitors, etc.), usually by means of a high-speed vision bus.

5.2.4. Compact Pyramid Prototypes

In the following, each of the four projects which have been introduced as representative of the compact pyramid family will be described in detail. The description focuses on the peculiarities of each system, which are compared only on a qualitative basis. A quantitative comparison based on real benchmarks is not possible because these machines at most reached the prototype stage and never became fully sized systems.

5.2.4.1. The PAPIA 1 System

The PAPIA 1 (*p*yramidal *a*rchitecture for *p*arallel *i*mage *a*nalysis)[2, 15, 16] is a common research project of a group of Italian universities begun in 1983, which reached the stage of a fully functioning prototype. This system belongs to the family of "compact" quad pyramids shown in Figure 3.14. The novel feature of the PAPIA system lies in the three-dimensional topology of the network of PEs included in the chip. In fact, it is based on a chip with five PEs (see Figure 5.6), which make up the basic two-level pyramid and which is the building block used to assemble larger pyramids.

PAPIA has a rather powerful PE (see Figure 5.7), since it is provided with two shift registers with programmable length and a complete arithmetic logic unit (ALU), and is therefore capable of implementing convolutions with kernels of arbitrary dimensions without having to save the intermediate results in the local memory. The main characteristics which define the performance of the machine are summarized below.

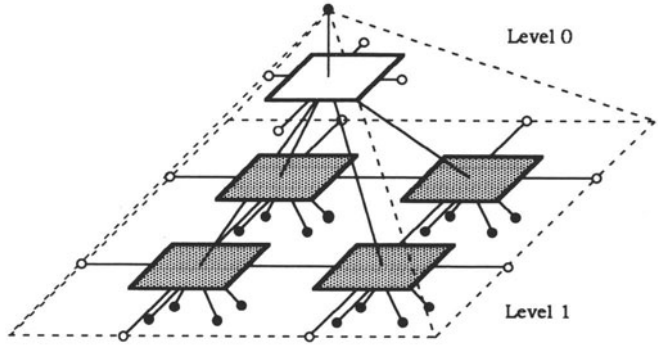


Figure 5.6. The PAPIA 1 elementary two-level pyramid of five PEs contained in a chip which constitutes the module for assembling the larger pyramid

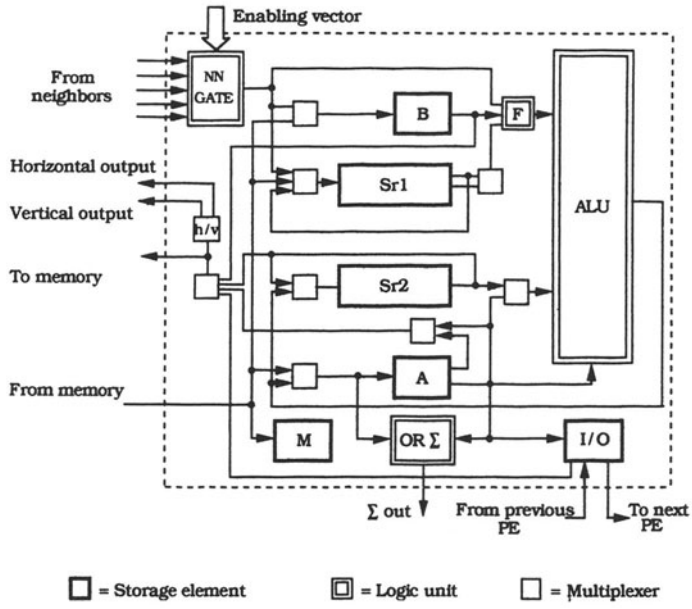


Figure 5.7. Block diagram of a PE of the PAPIA 1 pyramid. The two variable-length shift registers Sr1 and Sr2 increase the PEs processing capabilities in arithmetic window operations.

a. Interconnection Topology. The physical pyramid provides for direct access to intra- and infraplane neighbors and is a quad pyramid; each processor is directly connected to four children in the plane immediately below. The two modes of connection have been implemented in a disjointed manner: each processor can exchange data with its plane neighbors and with those on adjacent planes at different times. Two operative modes have thus been introduced even while maintaining homogeneity in the connections. These connections are the ''horizontal'' mode for exchanges between brothers, and the ''vertical'' one for communication between planes. This two-mode configuration was arrived at after observing that most algorithms which take advantage of multiresolution techniques operate in the two modes (vertical and horizontal) at separate times.

Communication between nonadjacent processors Communication between distant processors in fine-grained machines is difficult, since it has to be accomplished by adjacency via various intermediate steps. A second clock, which drives the transmission of data between one of the Boolean registers of the PE, was introduced in PAPIA in order to make these transfers quicker. Within certain limits, this allows the implementation of shift registers, with

variable topology, within the pyramid. This clock does not involve the ALU, and its frequency is a multiple of the frequency of the basic clock.

b. Processor Capabilities

Near-neighbor operations. A block diagram of the PE architecture is shown in Figure 5.7. Connectivity is 4 on the plane and is implemented with a broadcasting–gating solution: each processor distributes its own bit of information in parallel to its immediate neighbors. Data acquisition is via a gating circuitry (NN GATE in Figure 5.7) which collects the data coming from neighbors, with an equal enabling vector for every processor on the plane. The enabling vector is specified as part of the instruction code broadcasted to all PEs on a plane. In this way, each PE receives a signal which is the logical combination of the data transmitted by a subset of neighbors selected via the enabling vector. The logical functions carried out on the immediate neighbors are AND, OR, NAND, NOR.

Parallel access to immediate neighbors on the basis of a logical combination allows the recursive operative mode which is very efficient in the treatment of binary images. As described in Section 5.2.2.1b, by using the instructions in recursive mode it is possible to consider the connected components of an image, rather than the single pixel, as ''atomic'' data.

Memory organization. Two types of local memory are available: an on-chip serial access memory which constitutes the two shift registers for a total amount of 64 bits; and an off-chip random access memory which is 1 kbit per PE in the prototype. The external memory introduces some complexity to writing because chip operation is controlled by a single instruction, while the PEs can be individually masked off. Hence, external storage associated with a chip must be managed in one of two ways: (a) either with a read–modify–store cycle, in order to cope with the individual status of the PEs; (b) or with a write enable signal generated for each PE in the chip. The first solution, although more attractive in terms of pin count and board design, introduces a certain amount of complexity in the phasing of instruction execution, and therefore the second solution has been preferred.

The ALU. The ALU is made up of a serial adder, a serial comparator, and the circuitry that implements the NOT, AND, OR, and XOR logical operations. A carry register C, which is invisible from the outside, is considered part of the ALU.

The ALU is fed by two shift registers (Sr1 and Sr2), which are used to contain the data and the partial results in all the arithmetical and comparative operations. In the second type of operation, the third input from register A carries the partial result from the previous bit-serial comparison. As shown in

Figure 5.7, register Sr1 provides both normal and complemented output. The connections between the ALU, the interconnection to neighbors' logic, and the shift registers allow the execution of convolution operations with a kernel of arbitrary size without requiring that the partial results be saved in the local memory. An important characteristic of these registers is that they can be programmed in length with appropriate instruction. This simplifies the management of arithmetic–logic operations with a nonstandard number of bits per pixel. Moreover, the processor has been designed to simplify the serial implementation of multiplication, which is a basic step in convolution, by means of the logic unit F and because of the capability of the shift registers to rotate. In fact, unit F either acts as a multiplexer or it computes the AND between register B and the output from Sr1.

As well as the C register, two other single-bit registers have been introduced (A and B); their function is to accumulate the data and results of the logical operations or to be used as a support for arithmetical operations. All in all, the processing element is much more powerful than is the norm for fine-grained systems. A thorough analysis of the principles underlining the construction of such systems can be found in Fountain.[17]

Global sensors and recursivity. Each PE has an output line (Σ out); the lines emerging from the PEs which are on the same plane gather the state bit of each enabled PE (the A register) and are OR-ed together so as to collect information on the processing state of a plane of PEs. More precisely, it is possible to send either the content of register A or the difference (XOR) between its present content and past content on this output line. The first method allows an external controller to test for a global condition (i.e., all 0's or all 1's) in order to implement branching. The second method allows us to test whether there have been changes after the last instruction executed; this information permits the recursive application of the same instruction until a stable configuration is reached.

Operations of local and global propagation may be implemented by combining the technique of access to neighbors described above with this sensor: in the first case the result given by each processor is a function of the pixel contained in the local memory and that of the pixels contained in the selected adjacent processors (this type of instruction becomes useful in translations and in the search for particular templates in the 3×3 neighborhood of each pixel when implementing operators of mathematical morphology); in the second case the local propagation instruction is applied recursively, and the operation is ended only when propagation in the whole connected component is complete.

The instruction set. The 12 bits of the instruction code have been arranged into eight instruction classes (namely, memory access, shift register

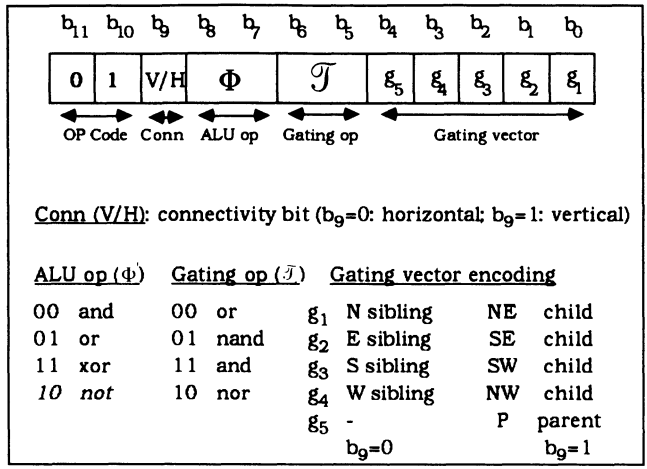


Figure 5.8. PAPIA 1 near-neighbor instruction format. The operation of the ALU and of the gating unit are detailed.

management, Boolean register management, arithmetic, logic, chip configuration, inter-PE movements, near-neighborhood logical); all in all, 58 operations are available. The *arithmetic* and *logic operations* are similar to analogous sets in other fine-grained parallel processors. In this case, however, direct support for bit-serial multiplication and comparison is provided. *Memory operations* can be used against registers A, B, the shift registers, the masking register, and the I/O distributed shift register.

The *chip configuration* class includes operations to set the length of the shift registers and to enable–disable either or both of the two planes contained in a chip. This latter feature allows time-multiplexed execution of different instruction streams at each level of the pyramid.

Inter-PE movements is a set of operations devoted to quick near-neighbor data transfer. Data movement between PEs is synchronized by the second clock of the system. To ensure better performance, only the NN gating circuitry and the Boolean registers involved in the transfer are active during the operation.

The format of the *near-neighbor operations* is depicted in Figure 5.8. Bits 11 and 10 identify the class. The operations come about by combining four gating functions $\mathcal{T}$ (bits 6–5) with the three Boolean operations Φ (bits 8–7) carried out by the ALU and adding NOT, which actually does not use the outcome of the gating unit. The decoding of the gating vector (bits 4–0) is conditioned by the mode specified by connectivity bit 9, which establishes either vertical (V) or horizontal processing (H). A near-neighbor operation in-

volves three registers in the PE: the status register A, which is the source and destination of the operation; the masking register M, which locally conditions the execution; and the propagation register B. As already anticipated, the near-neighbor operations come in two modes: local (nonpropagating) and recursive (propagating). Both execute as follows:

$$\begin{aligned} &\text{PE masked} \quad (M^0 = 0): \quad A^0_{t+1} = A^0_t \qquad\qquad \text{(data hold)} \\ &\text{PE active} \quad (M^0 = 1): \quad A^0_{t+1} = \Phi\,\{A^0_t,\ [B^0 \cap (\mathcal{T}^5_{i=1}\, g^i \cap A^i_t)]\} \end{aligned} \tag{5.5}$$

where superscripts refer to the neighboring positions, according to the encoding shown in Figure 5.8, with the further convention that 0 stands for the PE itself; subscripts refer to the execution step; Φ and $\mathcal{T}$ respectively denote ALU and gating circuitry operations. Equation (5.5) is the PAPIA 1 implementation of the general recursive NN operation of Eq. (5.2). The difference between local and recursive modes is in the number of execution steps: in the former, it is always 1, while in the latter it is data dependent, as established by the outcome of the global sensor. As an example of this mode, let us consider the connected component problem: an image, stored in register B, consists of more than one disconnected region; a single dot image (one pixel, the so-called seed, set to 1 and the others set to 0) identifies the component to be extracted and is stored in register A. The recursive execution of the following near-neighbor operation

0	1	0	0	0	0	0	0	1	1	1	1
		Hor	OR		OR		–	*W*	*S*	*E*	*N*
OP Code		Conn	ALU op		Gating op			Gating vector			

$$A^0_{t+1} = A^0_t \cup [B^0 \cap (A^1_t \cup A^2_t \cup A^3_t \cup A^4_t)] \tag{5.6}$$

propagates the seed (image *A*) by successive enlargements to cover the whole region of image *B* it initially identifies. The terminating condition of the recursion is

$$\mathrm{OR}_{\mathrm{image}}\,(A^0_{t+1}\ \mathrm{XOR}\ A^0_t) \neq 1 \tag{5.7}$$

and is detected by the external controller through the appropriate OR-Σout circuitry signals over the whole plane.

Local control. Of the three types of autonomy possible in fine-grained systems, PAPIA 1 supports operation autonomy only. Two levels of operation

autonomy have been introduced: a global, plane-level control and a local, processor-level control.

The first level permits parallel execution in the SIMD mode on a subset of arbitrary planes. Therefore operative modes are permitted which vary from SIMD over the whole pyramid to Multi-SIMD with as many processes as there are couples of planes.

The PE level, which is hierarchically dependent on the plane level, controls the individual processors. It permits the selection of arbitrary subsets of processors (within a plane or enabled planes) by loading appropriate enabling images into the M registers directly from the local memory or even using the partial results of a process.

c. System Configuration

Hardware system architecture. The basic hardware blocks of the first PAPIA system prototype, shown in Figure 5.9, are (i) a host computer (a SUN workstation); (ii) a controller[21] composed of a dedicated microprogrammed pyramid control unit (PCU) and a small general-purpose single-board computer (SBC); (iii) an image I/O subsystem containing a frame grabber and an active memory; (iv) the pyramid of the PEs.

The system is designed for a maximum configuration of eight levels and therefore has 128×128 processors at maximum resolution, i.e., at the pyramid base. In terms of chips, such a system would require 64×64 chips to assemble

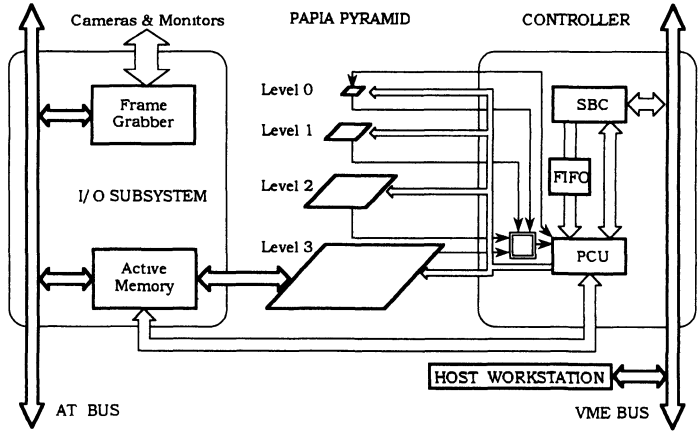


Figure 5.9. General view of the PAPIA 1 system. A mixed environment was used in this first prototype: the I/O subsystem is based on an AT bus, while the controller bus is VME.

levels 7 (128×128 PEs) and 6 (64×64 PEs); 16×16 chips to assemble levels 5 (32×32 PEs) and 4 (16×16 PEs); 4×4 chips for levels 3 (16×16 PEs) and 2 (4×4 PEs); and a last chip for levels 1 (2×2 PEs) and 0, the apex (containing a single PE). Thus, a total of 4367 chips is required for a system of that size.

The chip packaging, design methodology, and technology used in the project were those offered in 1987 by the silicon foundry (SGS/Thomson) as part of a nationwide multiproject chip program for universities. The design methodology used was a semicustom approach based on gate arrays. The libraries are based on a CMOS, 3 μm, double-metal process.

Analysis of chip complexity in terms of silicon exploitation gives an insight into the motivation for the choice of architecture. The PAPIA IC as a whole has a complexity of 17,000 transistors. The shift registers account for over 60% of the entire gate utilization. This is due to their considerable length (32 stages) and to the multiplexing logic applied to make them variable in length. Such a percentage of gates in the design is justified by the key role played by the shift registers which allow the circulation of partial results during arithmetic operations in local neighborhood processing. Since fine-grained architectures are generally best suited for low-level tasks, which rely heavily on near-neighborhood computations, the trade-off between cost and benefits can be justified.

A second, less evident aspect of the design is the effectiveness of Boolean, near-neighbor processing as carried out by the gating circuitry of the PEs data path. Both at PE and chip levels, such capability is obtained at a low cost, which in terms of equivalent gates does not exceed 4%. Since gated access to neighbor status allows the recursive processing and propagation which are qualifying features of the architecture, one can see that a crucial function in the system is obtained at a very low cost.

Pyramid prototype. The working prototype of the system is a four-level pyramid; thus, the base array consists of an 8×8 mesh of PEs. The 17 chips, which contain the 85 PEs, are allocated to five boards. The boards share the same layout and can accommodate a maximum of four chips; therefore, the one associated with the two uppermost levels of the pyramid only contains a single chip. The local memory of the PEs is obtained by a set of 20 1-kbit memory chips per board; support circuitry for memory management and drivers for high-fan-out wires complete the layout of these boards. A sixth board also exists that allows for the reconfiguration of the base of the pyramid, as far as edge connectivity is concerned. Three possibilities exist: with the first, the edges of the base array are set to a constant binary value, which is used in near-neighborhood operations performed by PEs on the periphery of the array;

the second one is the "wrap-around" configuration, through which North–South and East–West connections are established at the edges; the third option transforms the mesh of the base into a linear array by connecting the last PE of each row with the first one of the next row.

Loading and unloading of images. Image I/O is a bottleneck for all fine-grained machines: they all therefore require *ad hoc* hardware for the construction of bit planes from the images that are acquired in the form of a string of bytes.

In PAPIA an I/O register was inserted in every PE to allow the loading and the unloading of the images without interrupting task execution in the pyramid of PEs. The setup of these registers along the line is such that they make up an independent shift register driven by an external clock. The image is thus introduced by presenting the columns in sequence, bit plane by bit plane, and by making the shift register, whose length is equal to the length of the line, correspond to the row of the bit plane. In this way the image is loaded (and similarly unloaded) by making it shift with a new column for every clock cycle. It is then loaded altogether in the local memory once the pixels of the first column have crossed the whole image.

An external memory system called the active memory (AM) manages the transfer of the images through the I/O registers of the different PEs at the base plane of the pyramid.

The I/O subsystem was hosted, in the first prototype, by an AT personal computer. While the frame grabber is a commercially available board, the image reformatting unit, called active memory, was designed within the framework of this project. The functions carried out by the AM are transferring a window of the image stored in the frame grabber into its private storage and transmitting or receiving a bit plane of such a window to or from the pyramid I/O registers in columnwide format. The former procedure is initiated upon reception of a control word from the PCU, which sets the position and the dimension of the window to be transferred. Further synchronization allows the AM to start input–output operations with the base of the pyramid. The two activities can proceed in parallel. Moreover, the private storage of the AM is double ported, so concurrent transfer of bit planes to and from the pyramid is possible.

The controller. The control system of the PAPIA 1 is organized on three different levels: the microprogrammed unit PCU, the SBC, and the host. Instructions are broadcast to the pyramid by the PCU. All PEs in the same pyramid plane receive the same instruction at the same time. The PCU is directly connected to the pyramid of processors via dedicated, special-purpose links. The PCU is the custom-made, hardware controller of the pyramid; it holds the

program to be executed in the pyramid and generates all synchronization signals necessary both to the pyramid and to the I/O subsystem for image loading and unloading. It is built around a microsequencer, which drives a set of registers together with the microprogram storage. Each word of the control memory (80 bits) holds, among other control bits, the 12-bit op code to be sent to the pyramid. An extended set of microprogrammed pyramid routines is stored in the PCU memory as microsubroutines, accessible through short op codes from the SBC. While the interfaces to and from the pyramid are in the form of parallel ports, those to and from the SBC are FIFO queues, for better throughput and synchronization.

The SBC is interfaced with the PCU and with the Versabus Module Europe (VME)-based host. This second unit is a MC68000 SBC. It is in charge of the overall system synchronization and the execution of the scalar code of the applications. Indeed, even if the pyramid parallel unit is the target of most of the code, control structures embedded in the algorithms, such as selection and recursion, require the flexibility of a standard von Neumann machine. It is in such operations that the pyramid global sensor signal is used as feedback.

This description refers to the stated prototype; nevertheless the system is Multi-SIMD, and is capable of supporting multiple-instruction sequences on disjoint layers of the pyramid as long as different controllers are available to drive them. However, since every chip contains PEs of two adjacent planes and has just one instruction bus, only two planes sharing the same chips can execute the same instruction. Different couples of planes can simultaneously receive different instructions. When the pyramid operates in a vertical mode, the mode is SIMD over the whole pyramid due to the need to synchronize vertical transmissions.

Software environment. PAPIA has a multilevel programming environment, so writing and debugging programs is possible under varying degrees of difficulty and, correspondingly, of program efficiency.

The first level,[18] which allows for the complete visibility of pyramid architecture and its instruction set, involves programming the pyramid through microprogram development for the microprogrammed pyramid control unit (PCU). By microprogramming the PCU, the user can directly program and control the pyramid instruction stream sent by the PCU to the pyramid itself. A library of basic function modules (standard fixed and floating-point arithmetics, basic movement operations, local memory management, recursive binary and gray level near-neighbor operations, digital transforms, basic image processing modules, etc.) has been developed in this environment.

The second level consists of macroprogramming[19] the pyramid by means of the SBC. A set of functions (i.e., of pyramid instructions) is statically

loaded on the PCU, and the user develops programs for the SBC that activate these functions. This macroprogram development environment for the SBC is based upon an *ad hoc* macroassembler (PMACRO). As higher-level tools, moreover, a set of filters and procedures have been implemented that allow the writing of C programs for the SBC that contain macroassembly parts for the pyramid. Programs are written and complied on the host computer and are then loaded and executed on the SBC. The environment is completed with a simulator of the pyramid and of the PCU, which runs on the host computer and accepts PMACRO programs.

In comparison to the previous one, this environment allows for easier program debugging, particularly when complex macro functions for the pyramid are adopted as program building blocks, instead of functions composed of single instructions from the pyramid itself. On the other hand, lower system performance may results, because of the communication overhead between the SBC and PCU.

The third and highest level of programming is based upon PCL (Pyramid-C-Language[20]), a medium-level language designed to allow users to solve vision problems on pyramid machines. It is an extension of the C language, where parallel instructions and hierarchical data structures are included. Parallel data types have been introduced to represent basic pictorial objects and subsets of the pyramid array along with mechanisms for controlling concurrence. The PCL compiler is designed as a two-stage compiler: the first stage is a precompiler that translates a PCL source code into PMACRO code. The second stage allows us to separate the sequential components of the algorithms from the parallel ones and produces the executable code for the SBC linked to all the functions microprogrammed by the PCU. Details of this language will be given in Chapter 9.

5.2.4.2. The HCL Pyramid Prototype Machine

One of the first proposals for a compact pyramid architecture is the PCLIP system (pyramid cellular logic image processor)[21–23] developed by Tanimoto *et al.* at the University of Washington. The plan of this machine incorporates a set of cellular logic operators (CLOs) based on the quad-pyramid topology, as primitives of a pyramidal algebra of images called *hierarchical cellular logic* (HCL).[24] The machine is tailored for the implementation of these operators. The compact pyramid of PEs works in SIMD mode under the supervision of a single controller.

a. Interconnection Topology. The PCLIP is the only quad-pyramid project supporting the 8-connectivity on each plane in the hardware; i.e., the neighbor-

hood of a processor consists of 13 cells (see Figure 5.4): eight belong to the same level (lateral neighbors); five are hierarchical neighbors—that is, one located in the level above (parent) and four in the level below (children). With respect to the PAPIA design, in this case no distinction exists in accessing lateral or hierarchical neighbors; that is, parallel access to 13 cells is supported in the hardware to implement the CLOs.

In the prototype the chip integrates 4×4 lateral neighbors. In order to simplify the off-chip interconnections, the execution of the CLOs is decomposed into two phases, which allow us to precompute the partial CLO for the children and to complete it with the lateral and local cells. Details on these operations will be given subsequently.

b. Processor Capabilities. A block diagram of the PE architecture is shown in Figure 5.10. The architecture of the cellular processor is very simple. It consists of three single-bit registers, named P (''propagation''), L (''local''), and C (''condition''), and of matching circuitry to analyze any subset of the complete hierarchical neighborhood of a PE (''matching operation''). The P register contains the home data to be broadcast to the 13 near neighbors. The L register supports the dyadic operations. The C register is the enable register.

Near-neighbor operations. A unique feature of this architecture is that, except for setting and resetting of registers and for off-chip memory accesses, the only operation carried out is the CLO matching operation, in the AND__Match and OR__Match forms described in the following. The operations involve two operands: the first (EN) is the extended neighborhood which consists of the P registers of the processor itself and of its 13 neighboring processors plus the data stored in the local register L. The second operand (MP) is a pattern broadcast from the controller which describes the local structuring element: each of the 15 extended neighborhood values is in ordered correspondence with one digit in the pattern specifying the 0 or 1 or *D* (don't care condition) value. The definition of the AND__Match operator is

$$\text{AND\_\_Match} = \bigcap_{i=0}^{14} (\text{MP}_i \diamondsuit \text{EN}_i) \tag{5.8}$$

where $\text{MP}_i \diamondsuit \text{EN}_i$ is equal to 1 if $\text{MP}_i = \text{EN}_i$ or $\text{MP}_i = D$; otherwise it is 0 (the extended neighborhood labeling convention is the one shown in Figure 5.4, with the home data labeled, respectively, 0 for register P and 14 for register L). That is, the result is 1 if each and every element of the pattern matches its associated element in the extended neighborhood, or it is 0 when at least one

element does not match. This operator is equivalent to the "hit-or-miss" operator of mathematical morphology,[25] in which the two disjoint components of the structuring element are a subset of EN and the remaining points of EN are associated with D.

The definition of the OR__Match operator is

$$\text{OR\_\_Match} = \bigcup_{i=0}^{14} (\text{MP}_i \dotplus \text{EN}_i) \tag{5.9}$$

where $\text{MP}_i \dotplus \text{EN}_i$ is equal to 1 if $\text{MP}_i = \text{EN}_i$ and $\text{MP}_i \neq D$; otherwise it is 0. That is, the result is 1 if at least one element of the pattern different from D matches its associated element in the extended neighborhood. It is 0 when no element matches.

The two Match operations are the PCLIP machine primitives. Arithmetic, logical, and data movement operations must be implemented through them. For example, shifting a bit plane, stored in the propagation registers, by one position can be achieved with one of the following operations:

$$\text{ShiftWest(P)} = \text{AND\_\_Match}(D1DDDDDDDDDDDD, \text{P}) \tag{5.10}$$
$$\text{ShiftWest(P)} = \text{OR\_\_Match}(D1DDDDDDDDDDDD, \text{P}) \tag{5.11}$$

The digit set to 1 in the pattern corresponds to the eastern neighbor according to Figure 5.4; its value is copied in the home register P, with the remaining data in the neighborhood masked off by the "don't care" conditions.

Memory organization. An off-chip static random access memory, 8 kbit per PE, is available in the prototype. This memory is physically wired to the vertical transmission line to the children and is driven by an appropriate tristate logic. The whole base-level memory is shared with the host computer; the arbitration criterion is based on a nonpreemptive priority mechanism which favors the host CPU. Moreover, during the system configuration phase two bit planes of the base memory are set to the constant values 0 and 1, respectively, and are used to feed the base with the dummy values needed for the matching operations.

The condition register C cannot mask memory operations, so selective writing must be implemented with a read–modify–store cycle.

Arithmetic and logical operations. These operations are executed on data stored in registers L and P. In particular, AND, OR, and NOT operators are a subset of the matching operations, and the other two input Boolean functions can be implemented on the basis of these two primitives.

Multibit operations are implemented with the two basic primitives as well; they require, in general, several cycles per bit: in fact, the hardware available is confined to a two-input, single-output matching circuitry. For example, adding two single-byte data requires 226 machine cycles[26]: for each bit two XOR, two AND, and one OR operations are needed, plus a number of intermediate results storing and loading operations, giving a total of 28 cycles.

Local control and instruction set. The only local autonomy introduced is the operation autonomy as supplied by the one-bit *C* register.

The 18-bit instruction set of the PCLIP is extremely simple, and, besides the matching instructions, consists of the SET and CLEAR operations and LOAD and STORE operations. Both of these groups act on the three single-bit registers P, L, and C.

The unconditioned STORE operation produces as a second result the OR-of-all signal (see Figure 5.10) which at chip level collects the inclusive OR of the 16 registers involved in the operation. This signal is used to build a global feedback for the controller and can be exploited for testing, for branching, and in the implementation of recursive operations to reveal whether a stable status has been reached in the array.

c. System Configuration

Hardware system architecture. The basic hardware blocks of the PCLIP system prototype are shown in Figure 5.11: (i) a host computer IBM PC/AT in charge of the overall control and of the image I/O subsystem; (ii) a simple controller containing the instruction memory, the sequencing logic, and the interface to the host; and (iii) the pyramid of PEs.

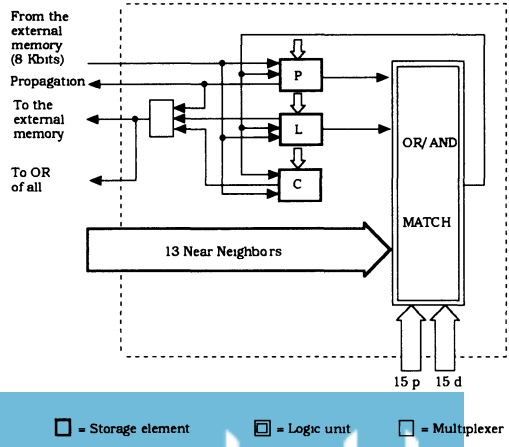


Figure 5.10. Block diagram of the PE of the PCLIP pyramid Note that the PE is reduced to the minimum set of units necessary to implement the basic CLOs by an effective broadcasting–gating technique

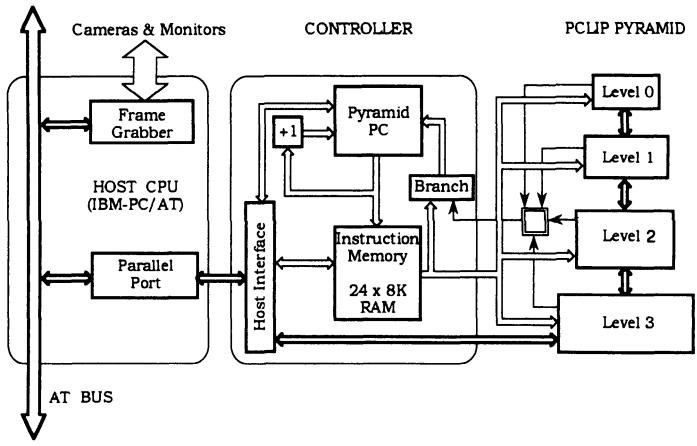


Figure 5.11. The PCLIP pyramid prototype. The whole pyramid system can be seen as an attached processor, easily interfaced through a standard parallel port.

The designers of PCLIP have investigated possible implementation strategies with VLSI technology to cope with the tremendous problem of interconnecting the processors; among these are recursive layout and time multiplexing of interconnections,[21] but the final prototype relies on a chip implementation which will be presented afterward, based on PE interconnection peculiarities. Common to other projects is the provision for a single chip to be used in the whole machine.

The chip containing a planar 4×4 array of PEs has been integrated using a custom approach in NMOS technology and has a 64-pin dual-in-line package. It would be a simplistic approach to conceive the PEs as a self-contained unit; the silicon area has been better used by "grouping together in one unit all one element pattern matchers that share a two-bit pattern symbol."[22] In this way, the number and the routing length of the control signals is minimized.

Matching is the only computation carried out by the processor. Since it can be logically decomposed into two suboperations, one building the vertical matching and the other extending the result with the inclusion of lateral matching, it can be implemented at chip level by precomputing the value of each 2×2 subarray of PEs, thus needing only a single connection from the four children to their parent. Despite this reduction in the number of connections, the pins available with the package are insufficient to bring the whole 30 bits of the pattern into the chip, together with the 40 bits of the near neighbors (20 lateral and 20 vertical). Therefore the matching computation is executed as a

sequence of two phases, taking note of the decomposition described above and of the number of pins shared to load the complete pattern.

The chip supports a reconfiguration function to mold the apex and the 2×2 level of the pyramid into a single device.

The pyramid prototype. The pyramid prototype consists of four levels with a base size of 8×8. The six chips which contain the 85 PEs are placed on two boards: one for the base (four chips) and the second for the three higher levels (two chips). Both boards host the external memory of the PEs (64×8 kbits and 21 kbits, respectively). A single cage contains the pyramid and two more boards for the controller.

The controller. The two controller boards consist of the pyramid program memory (8K instructions 24 bits long), pyramid clocking, interface toward the host, and instruction sequencing logic.

The controller instructions can be grouped into the following types: pyramid types combined with the PE external memory addressing; NOP and HALT operations (the former necessary at the initialization of the system, the latter to pass the control to the host CPU); and branch operations, which can be modified using the output of the OR-of-all signal.

The host interface consists of a set of 12 registers accessible through the parallel port. Among these registers are a control and status register, used to halt the controller and to initialize the program counter; two registers to specify the instruction address and three to transfer the instruction data; two registers to specify the external memory address; and three registers used to manage the I/O communication between the host and the base of the pyramid.

The image loading–unloading. The mechanism of exchanging image data with the pyramid is very simple. As mentioned, the external memory of the base is accessible from the host, when this has priority over the PE pyramid. In particular, each bit plane of 64 bits can be addressed as a set of bytes, with the 8 bits of each byte corresponding to a single row and eight different columns. The addressing of each byte is obtained through the three registers of the host interface as follows: two registers specify the position (row, column) of the byte within the array of PEs; the third contains the actual byte of data to be read and written.

5.2.4.3. The GAM System

In the acronym GAM, G stands for *G*eorge Mason University, A stands for a supplementary pyramid of adders, and M stands for MPP (massively parallel processor).[4, 27] The pyramid system built at George Mason University by Schaefer *et al.* has the feature of being assembled out of existing microcir-

cuits of the MPP from NASA,[28] augmented with a tree of adders for fast bit counting.[29]

a. Interconnection Topology. The GAM pyramid has been conceived as a standard nonoverlapped, 4-connected quad pyramid, with the complete neighborhood of a PE consisting of four children, four brothers, and a parent. However, unlike the other compact pyramid systems, GAM has been assembled using an integrated circuit originally designed for a flat processor array (MPP), not made for a hierarchically interconnected network. The major design effort therefore has been to devise a way to use the MPP chips, each containing a 2×4 array of PEs, to be used as the building block for the pyramid.

Specifically, while the MPP chip supports lateral near-neighbor interconnections, it lacks hierarchical links; yet it offers eight bidirectional data channels for memory operations that can be used for any type of transfer. Parent–children data exchanges have been implemented over these lines by properly interposing a tristate circuitry at each line: between external memory chips, between the chip containing the parent PE, and the one containing the children PEs. The control signals driving these circuits enable a monodirectional data transfer between two adjacent layers of the pyramid. Logically, the tristate circuitry performs a *replicator* function when broadcasting data toward the four children and a *selector* function when accessing data out of the four data inputs.

b. Processor Capabilities. Since the GAM system relies on the MPP chip, the capabilities of the processing element (see Figure 5.12) are described here only briefly. Details can be found in Batcher.[28]

Near-neighbor operation. Access to the neighboring PEs at the same level is through a multiplexing technique: the data available in the P register are broadcast outside the PE and are loaded into the P register from one of the four inputs of the NEWS communication network. Vertical communications are implemented as memory read operations, with the external tristate circuitry providing the correct data from the single PE of the adjacent level involved in the transfer. Thus, a mixed-mode operation is executed, with decoding carried out partially at the PE level and partially at board level.

Memory organization. Thirty-two bits of on-chip local memory are available in the form of a variable-length, serial access shift register. In the GAM system, every PE has been equipped with 8 kbits of external memory; storing into this memory is not conditioned by the enable register G, so a read–modify–store cycle is used.

Arithmetic and logical operations. Two units in the PE take care of ar-

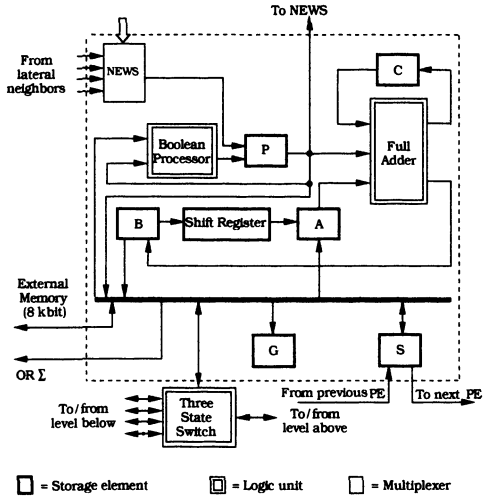


Figure 5.12. The processing node of the GAM pyramid. The structure of the MPP elementary processor is enriched by the three-state switch for vertical communications. The arithmetic and logic capabilities of the PE derive from the Boolean processor, the full adder, and the variable-length shift register.

ithmetical and logical operations respectively: the former is a standard bit-serial full adder, whose inputs are from the first bit of the shift register (acting as the A register), from the near-neighbor register P and from the carry register C; the latter unit only executes one of the Boolean operations on the contents of register P and on the data available in the data bus, while the result is stored in register P. The 32-bit, variable-length shift register is the recipient of arithmetic operations in its last bit (acting as register B) and supports multibit operations.

Local control and global sensor. Register G is the enable register: its content, combined with a bit of the transmitted instruction, establishes whether the PE stores the results of the operation carried out or keeps its previous value. The sum-OR signal, used to provide the controller with a feedback from the pyramid for testing and branching, is obtained by ORing at board level the values available on the data bus of the PEs.

c. System Configuration

Hardware system prototype. The GAM working prototype consists of four major units: (i) five-level pyramid; (ii) tree of adders; (iii) controller–host CPU; and (iv) a dedicated I/O subsystem (see Figure 5.13).

The main characteristics of the MPP chip are summarized here briefly. Containing an array of eight PEs arranged as a 2×4 mesh, the chip was fabricated using a mixed CMOS–NMOS technology: the faster NMOS process was used to implement the data bus interconnecting the storage elements with the ALU and the logic circuitry.

Pyramid prototype. The five-level pyramid of 341 PEs was assembled with 44 MPP chips laid out on 13 boards. Each board hosts a "module" of four MPP chips—an array of 4×8 PEs. The base of the pyramid (16×16 PEs) is made using eight modules, the 8×8 level using two modules, while the remaining three topmost levels each use a partially configured module to embed the 4×4, 2×2, and 1×1 arrays (respectively with two, one, and one chips).

Besides the four MPP chips, each board contains the following units: the external memory for the PEs, with four 8-kbyte chips providing each PE with an 8-kbit address space; and the off-the-shelf tristate circuitry for hierarchical interconnections and additional discrete components for the sum-OR function.

Tree of adders. The level above the base, which contains an 8×8 array of PEs, communicates with the controller through a special counting subunit, a tree of adders: this is capable of producing the 7 bits representing the sum of a bit collected from all 64 PEs on the level, within 100 nsec after its inputs are stable. Considering that the clock cycle of the MPP chip is exactly 100 ns, it is possible to count all the PEs set in the base within a time period equivalent to eight clock cycles: indeed, by selectively loading the PEs at the 8×8 level with the data from their children and activating the counting circuitry, four vertical accesses plus four counting operations complete the sum. This is a considerable gain over the 178 cycles necessary to perform the same operation with a standard recursive doubling technique within the base. The tree of adders occupies one of the 15 boards in the cage of the prototype.

The controlling environment. The GAM pyramid is controlled by an AT&T IBM-compatible personal computer, acting both as the controller and as

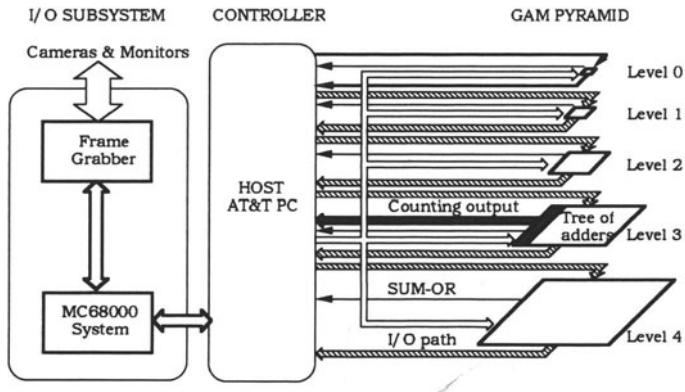


Figure 5.13. The simplified block diagram of the GAM pyramid. Note that the I/O parallel paths are available to every level. Only level 8×8 has the tree of adders counting subsystem.

the host computer. In its controller function, it drives the 44 lines that bring the instructions broadcast and the clock signals to the boards, and collects the 43 lines emerging from the pyramid carrying the signals for the sum OR (5 of them), the tree of adders output (7 of them), and the I/O paths (16 of them in the base, then 8, 4, 2, and 1 in the upper levels).

The mode of operation of the pyramid is SIMD for all operations except vertical transfers. To implement these transfers, the controller issues a special instruction that specifies the couple of planes involved in the data exchange, the role of each plane either as the sender or the receiver, and the proper memory access instruction for the PEs. Thus, the vertical transmission obeys a senser–receiver protocol, whose synchronization at board level prevents the execution of more concurrent vertical transfers among planes.

The host CPU interfaces the pyramid with the I/O subsystem. Special circuitry designed around a Motorola MC68000 microprocessor performs the video signal reformatting operation and prepares the 16×16 array of bits suitable for column parallel loading into the distributed shift register S in the PEs of the base. The I/O channel can be routed to the other planes of the pyramid as well.

The GAM II enlarged pyramid. Recently, the first prototype has been expanded by adding a sixth layer. The GAM II machine now has a base of 32×32 PEs, built with 32 more boards. The resulting 45 boards interconnect through a back plane that contains three separate buses. One of the buses drives the base, another the level above the base, and the third the rest of pyramid. At the time of this writing, the three control buses deliver the same instruction to the pyramid, which therefore remains a SIMD machine.

5.2.4.4 The SPHINX System

The Système Pyramid Hiérarchisé pour le traitement d'Image Numériques (SPHINX) was conceived by a joint effort of the Universitè de Paris-Sud and the ETCA defence research laboratory.[30, 31] This system belongs to the family of ''compact'' bin pyramids, see Figure 3.14. The pyramid structure is characterized by intralevel interconnections arranged as a 4-connected mesh (for horizontal communications) and interlevel interconnections in which PEs can communicate on a vertical binary tree. Therefore the PEs are alternatively the father of two sons having the same X coordinate or the same Y coordinate. The way in which the binary tree is embedded characterizes the hierarchical structure of the systems; the internal PE organization mirrors this duality since there is an equivalent circuitry on both vertical paths. The SPHINX system therefore is well suited to dealing with tasks relying on binary tree primitives.

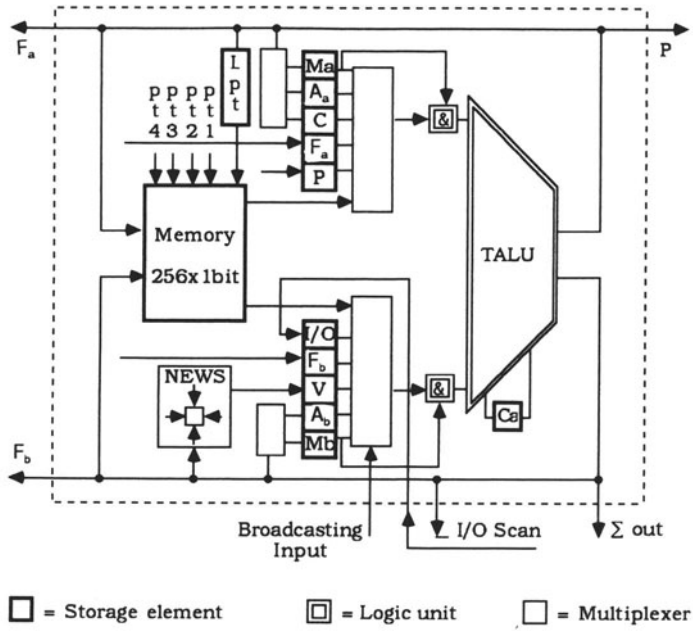


Figure 5.14. Block diagram of a PE of the SPHINX pyramid. Note that the dual structure of the PE (horizontal symmetry of the scheme) mirrors the bin topology.

a. Interconnection Topology. Although the degree of each node in the bin pyramid is lower than in the quad pyramid, the latter structure is more ''isotropic.'' The number of PEs is greater in the bin pyramid: $2^n \times 2^n$ being the base size, the total number of PEs is $2^{2n+1}-1$ and $(2^{2n+2}-1)/3$, respectively, for bin pyramid and quad pyramid. Finally, the number of layers is smaller for the quad pyramid: $2n+1$ and $n+1$ respectively for bin or quad pyramids. This also means that the number of stages to be passed through is almost double when computations are going from the base to the apex or when communications between nonadjacent PEs are implemented via the shortest vertical path.

b. Processor Capabilities

Near-neighbor operations. A block diagram of the PE architecture is shown in Figure 5.14. The circuitry is divided in two: the upper part is related to data exchange with the (a) child and, symmetrically, the lower part with the (b) child. The connection to the parent requires the same circuitry as with the

first child, while the brother (NEWS input) data access requires that belonging to the second child (as a consequence, while data coming from both children can be processed simultaneously, mixed operations have restrictions: the brothers can be combined in a single operation with the parent or the (a) child; the parent with one of the brothers or with the (b) child). Moreover the neighbor data access is implemented with multiplexing for intralevel communications (NEWS access) and by parallel mailboxes for the interlevel data exchanges. Since vertical paths are bidirectional and the communication primitive is *read,* vertical communications must be synchronized by the external controller. Nevertheless a minimal degree of autonomy in managing the vertical channel is supplied by a pair of single-bit registers which can mask off the data read from the children. This feature allows the choice of an arbitrary subset of the links of the binary tree and is one of the distinctive features of the SPHINX machine. Besides, since the NEWS logic is based on multiplexing, the near-neighbor recursive propagation is not possible.

Memory organization. The local memory is divided in two: four banks of on-chip static RAM (1 kbit per bank yielding 256 bits for each of the 16 PEs in the chip) and an off-chip RAM (8 kbit per PE in the prototype) accessible through the parent data link. From the logical point of view the on-chip RAM can be considered a dual-input, dual-output device consistent with the symmetric PE data path.

On-chip memory can be addressed through five pointers: four of them (pt1–pt4 in Figure 5.14), loaded by the external controller, supply a direct implementation to the four-way addressing modality of the RAM and allow an effective looping implementation by means of an autoincrementing circuitry (these four pointer registers have been replicated on each chip in order to reduce address transmission between controller and PEs); the last pointer (Lpt) is local to each PE and is loaded by Transfer ALU (TALU), thus giving rise to local addressing autonomy (this pointer lacks autoincrementing capability and can only be processed serially through the TALU).

Besides feeding the TALU in local operations, the on-chip memory can be broadcast to the NEWS network for near-neighbor operations.

The TALU. The design of the PE data path is conceived on the basis of a *read–modify–store* cycle. The TALU is fed by two symmetrical sets (a and b) of Boolean registers. Each set has a general-purpose accumulator register (A), a masking register (M) whose content is ANDed with the TALU input, a child register (F), and a second near-neighbor register (either the parent P or the NEWS result V). The set (a) furthermore includes the activity bit (C) which makes the PE conditionally execute the current instruction. The set (b) includes

an I/O bit which is used to build a fast path for image input–output. Furthermore the two TALU inputs can be fed directly by the on-chip memory and then by the input broadcast to all PEs by the controller.

As well as two inputs, the TALU also has two outputs which almost symmetrically can reach both halves of the PE. The common destinations are the on-chip memory, the general-purpose accumulator registers (A), the masking registers (M), and the children channels (F). On the difference side, output (a) only can load the local pointer (Lpt) and the condition register (C) and transmit data to the parent channel (P); output (b) is the only one to be routed to the brothers (through the NEWS circuitry), to the I/O scan path, and to the global OR (Σout).

The TALU contains a logical unit, an arithmetical section equipped with a carry register (Ca), and a switching circuitry for data distribution that allows us to route the internal results to both outputs. As the name suggest the inputs can be directly switched to the output.

Local control. All three types of autonomy available in fine-grained systems, described in Section 5.2.2.4, are supported, even if partially, in SPHINX systems.

Operation autonomy is implemented through registers C, Ma, Mb, and Ca. In addition, the instruction code specifies whether the execution is to be conditioned by the C register, which acts as the enabling register described in Section 5.2.2.4. The ANDing function executed on the inputs to the TALU by the M registers modifies, on the basis of local outcomes, the operands, thus causing different behaviors from the PEs. Moreover, the Ca register can operate as a switching element for the swapping of the two TALU outputs, thus realizing the following directive data-moving instruction:

$$\begin{aligned} \text{Out\_}a &= (\text{In\_}a \cap \text{Ca}) \cup (\text{In\_}b \cap \overline{\text{Ca}}) \\ \text{Out\_}b &= (\text{In\_}b \cap \text{Ca}) \cap (\text{In\_}a \cap \overline{\text{Ca}}) \end{aligned} \tag{5.12}$$

Addressing autonomy is supplied by the local pointer (Lpt), managed by the TALU, and is active only on the on-chip memory as previously described.

Connection autonomy is implemented by locally gating through the M registers the data coming from the neighbors before feeding them to the TALU.

c. System Configuration

Hardware system architecture. The SPHINX pyramid system will consist of four major subunits, organized around a VME bus: (i) the PE pyramid;

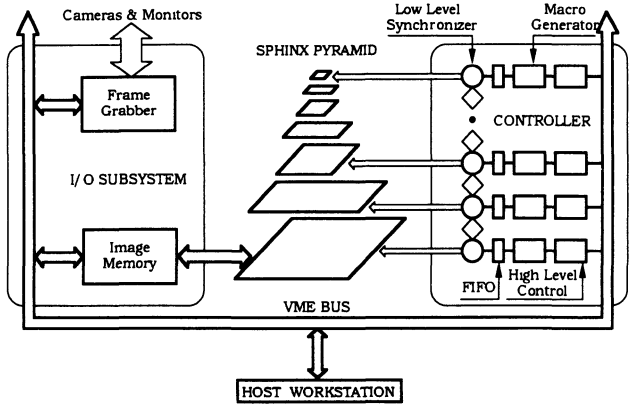


Figure 5.15. The structure of the SPHINX pyramid prototype. Starting from the 1024 PEs of the base the remaining 10 layers are assembled with 1023 more PEs.

(ii) a set of plane controllers; (iii) a dedicated image memory board; and (iv) the host computer (see Figure 5.15). Currently, a first prototype is in construction, and the following description refers to it.

The pyramid prototype. The PE pyramid has a 32×32 base, resulting in an 11-level structure made up of 2047 PEs. Since each chip hosts a 4×4 subarray, the first seven levels of the pyramid require 127 chips, each of them fully exploited. The last four stages, if actually assembled, would require four more chips to complete the pyramid, but each chip would be used only partially, since levels 8, 9, 10, and 11 consist respectively of 2×4, 2×2, 1×2, and 1×1 PEs. To make this possible, the chip contains a special reconfiguration circuitry that adjusts the interconnections properly.

The chip is designed with CMOS 1.5-μm gate-array technology and has been fabricated by VLSI TECHNOLOGY.[32] Sixteen PEs are integrated in an 8×8 mm^2 die for a total of 100K transistors, including the 256 bits per PE of on-chip memory. Its basic clock cycle is 10 MHz.

The prototype is being fabricated in two stages: in the first stage, it will embed a small pyramid, with 64 PEs in the base; in the final configuration, the 32×32 base will be constructed in a modular way to allow for further extensions.

The small pyramid consists of two extended double-Eurocard boards. Each board hosts a maximum of six chips. Each chip has access to 16 kbytes of external memory that are organized as 16 separate 8-kbit address spaces, one for each PE in the chip. The boards for the enlarged pyramid will be

twice as large as those for the small pyramid and will accommodate up to 16 chips.

Image loading and unloading. The pyramid does not interface directly with the VME bus for image loading and unloading, but connects instead with a special I/O board which acts as a staging memory. The main task of this board will be the reformatting of the image data for proper transmission to the base of the pyramid. The parallel I/O path to the pyramid is based on the single-input–single-output *scan path* available in each chip and shared within the chip by the 16 PEs. So, each of the 32 streams of data flowing toward the base consists of packets of bits organized as four nibbles and carrying the correct sequence of 16 bits for the PEs of each chip.

The controller and the host. The controlling subsystem of the prototype is a major design effort.[33] Since the SPHINX pyramid is designed to operate with a Multi-SIMD control structure, each level of the pyramid requires a dedicated controller, responsible for the task running on its layer and for the synchronization of data exchanges with the two adjacent layers. The functional structure of each controller is depicted in Figure 5.15. A high-level control section, based on a standard microprocessor, interfaces with the host through the VME bus, receives the code to be executed by the PEs of the layer and manages the scalar code and data of the task. A second unit is the macro generator: its purpose is to expand the instruction transmitted by the host, which operates on multibit quantities, into the rather long sequence of micro-operations actually executed by the PEs. The third unit, a FIFO queue, receives such microinstructions which are later transmitted to the layer: the FIFO decouples the two types of activities (writing for the controller, reading for the PEs) and allows an asynchronous mode of operation between the controller and the PEs.

The synchronization of the tasks running on adjacent layers of the pyramid is the critical part of the Multi-SIMD control strategy of SPHINX. Different approaches have been proposed, which are based on alternative strategies of programming the pyramid to execute cooperating processes. A thorough analysis of the implications on the programming models of the synchronization process is looked at in Chapter 9. In this context it suffices to highlight the hardware resources on which the synchronization mechanism is built. Data exchanges between two adjacent layers are synchronized by creating named channels in the off-chip memory, which are shared between the two layers. The low-level synchronizer depicted in Figure 5.15 is the minimal hardware required to solve contention on the access to the shared memory. It is the responsibility of the tasks running on the layer to issue the proper instructions to activate the synchronization policy when interlayer transmission is required.

The prototype system cannot work as a stand-alone machine but requires a workstation both for program development and execution. Any VME-based machine is suitable as a host; in the first prototype a SUN 4 system was used.

5.3. Distributed Pyramids

Compact pyramid architectures, such as those examined so far, fall within the fine-grain paradigm and build three-dimensional machines by assembling specially designed integrated circuits. Each of these embeds more than a single computational node. The class to be examined here follows the complementary, coarse-grain approach: each node is capable of quite advanced functions and is usually composed of a standard microprocessor, which accesses a bank of high-speed standard memory. It follows that the number of nodes which make up a prototype is much smaller than that found in compact pyramids; also, motivations for the systems and practical realization problems are quite distinct.

If one examines the image processing and computer vision domain, the term *distributed* is easily justified: in the early and intermediate stages of the computations, such systems process blocks of data on a per-node basis rather than associating each node with a pixel of the image. To do so, the available nodes are scattered and properly distributed over the image data. As mentioned, this type of processors-to-image allocation favors computational-intensive, local operations whose communication graph is liable to stay completely within the image block local to the processor. Instead the recursive- and propagation-type operations are more difficult because of the border transition problem which ensues when neighboring data reside on memory blocks of different processors. However, if the algorithms to be executed move into the medium- to high-level domain, the more flexible data structures necessary for the computations are better mapped in the block-partitioned memory of distributed systems rather than in the tightly orchestrated one of compact systems.

Moreover, computer vision is not the only application domain where the coarse-grain approach can be used efficiently. Actually, distributed, hierarchical systems have been called upon largely for computationally intensive, data-parallel computations[34] such as numerical simulation of physical phenomena, molecular dynamics, Monte Carlo methods, large-scale integrated circuit physical simulation, computational chemistry, and computational fluid dynamics. These problems map very well onto a homogeneous, parallel architecture because the underlining physical phenomenon is ''space extensive'' and can be modeled with local interactions. A gridlike interconnection of powerful processing nodes does the job of numerical simulation very well and can often be

added to with a limited number of additional, reduced resolution grids in order to implement multigrid algorithms.

The architectural requirements of such systems therefore constitute the embedding of powerful, potentially autonomous and self-supporting microprocessors into a regular topology network. The case analyzed here is that of pyramid topology. The interconnection of such processors in a hierarchy poses implementation alternatives which are quite different from those of compact systems. The processors are usually standard components; as such they do not come with dedicated interconnections ports (the transputer of INMOS is the exception), but are rich in communication facilities. Parallel and serial I/O ports are the first possibility; the transfer rates that can be obtained are, however, quite limited, and the associated protocols rather cumbersome. A second strategy is memory sharing; it allows for large common address space between neighboring nodes, but is somewhat difficult to manage in the synchronization of accesses. Dual-ported devices are becoming quite common, thus offering a basis for the implementation of virtual links between two processors. If the architecture requires it, special-purpose, multiport controllers can be designed to augment the interconnectivity of the processor–memory module.

A final characteristic of distributed pyramids stems from the autonomy of operation conceptually available at each node. A standard microprocessor must fetch its instructions from a local memory and runs the resulting algorithm with little or no interaction with a controlling device. Yet, the overall structure of the systems suggests two possible ways to use the inherently hierarchical arrangements of processing nodes: either according to a multiresolution approach where all layers process data at different resolutions or according to a ''worker–dispatcher'' paradigm where one layer (the bottom one) takes care of the actual computations and the upper ones execute administration management tasks, such as program and data downloading, task dispatching, and I/O management. Roughly speaking, it is possible to conceive a hierarchical distribution of the algorithms, where the control portion is mapped at higher levels of the hierarchy, with the base performing only numerically intensive activities. While such a paradigm offers a distinctive implementation of the Multi-SIMD control strategy, it entails a much higher level of granularity than is possible with compact systems.

5.3.1. The EGPA System

A notable embodiment of the concepts that have just been described is the set of prototype machines[34–36] which can be collectively referred to as EGPA (Erlangen general-purpose array). The EGPA program is based on a number of

functions, briefly described here, which have led to the construction of two fully functioning prototypes.

a. Interconnection Topology. The systems, built or conceived, follow the quad-pyramid topology; however, while the processors of a layer are interconnected in square tessellation through bidirectional, symmetrical channels, hierarchical communications are monodirectional and place the burden of activating and managing the communication on the parents.

b. Processing Node Capabilities. The EGPA program is based on a distributed, shared-memory approach. Each node is composed of a processing unit (a standard microprocessor) and a memory module and is referred to as a processor memory module (PMM). The processing capabilities of a node remain with the microprocessor; however, sometimes the standard instruction set of the microprocessor could be extended by microprogrammable attached processors that can be best tailored to the target computation. The interconnections between nodes are obtained by sharing memory modules. To obtain the degree necessary for a quad-pyramid topology, such memory sharing is achieved by custom devices acting as multiport memory controllers.

The overall approach to setting up the node of an EGPA system is that of following a building-block philosophy: a node can be used to set up systems according to the topology that best matches the tasks, and sufficient interconnections are available for high-degree topologies, particularly for quad pyramids.

c. Control Structure. The hierarchy of planes reflects the processing hierarchy typical of numerically intensive data-parallel computations. The base level of the hierarchy acts as a ''worker's'' array, activated and controlled by a second layer of controllers and dispatchers, while the overall control of the system is shared by the topmost node and the host. The systems are designed to operate in a SPHD mode as far as operations on data are concerned: the array of ''worker'' nodes executes the same program, which has been loaded in their private memory by the parent node by accessing their shared memory.

d. Expandability. The use of standard microprocessors and memory modules and the resulting architecture of the node lay the foundation for system expandability: the only limitation to system expansion and reconfiguration is due to the mechanism by which memory is shared. The multiport approach in the largest prototype supports a maximum of eight connections.

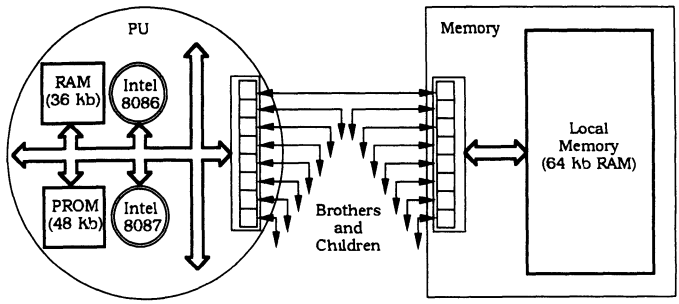


Figure 5.16. Block diagram of the PMM node of the DIRMU pyramid prototype. The 16 unidirectional links are subdivided into eight outputs from the PU, and eight inputs to the local memory module of which only five are used to set up the pyramid topology.

e. The Prototypes. The EGPA pilot prototype[35] was the first realization of the EGPA program, built and operated between 1977 and 1983. It is a two-level pyramid assembled with five PMMs. Each node contained an AEG 80-60 processor: its main features were a 32-bit architecture and user microprogrammability. The pilot pyramid operated under the control of a standard operating system adapted to the hierarchical environment: mailbox techniques for tasks communication, data partitioning, and task allocation–deallocation were among the chores demanded of the operating system.

The DIRMU subprogram[34] led, in 1985, to the replacement of the pilot pyramid with a three-level pyramid. The processing node of the PMM was assembled with 8086/8087 INTEL microprocessors (see Figure 5.16): locally and privately to the processors were 36K of RAM and 48K of ROM memory, which hosted local components of the operating system and self-test programs. Connectivity with other PMMs was through a maximum of eight channels for memory accesses. The memory module contained a multiport, an eight-way control device, which controlled access to 64K of standard memory.

Larger EGPA systems have also been conceived as well. The EMSY 85 pyramid[36] was designed to reach a four-level structure: the prospective processing module was to contain an INTEl 286/287 couple, while the memory module would allow for a half-megabyte of local addressability.

The largest configuration[35] envisages a 16×16 base-level pyramid, with a node specified to offer a conventional 32-bit advanced mirroprocessor (either Motorola 68030 or INTEL 80386) with 4 Mbytes of multiported memory, driven by a special-purpose controlling device and with an attached high-speed microprogrammable coprocessor for special-purpose instructions. This system was also conceived according to the "worker–dispatcher" mode: the 8×8

level hosted the distributed operative systems, with 64 specially designed I/O channels for high-speed disk access (10 Mbytes/sec transfer rate each). The prospective pyramid had one more level, instead of three more, consisting of a single PMM node linked by a special bus to the PMMs at the 8×8 level: it was therefore a flattened structure.

The specification of the system gives rise to an overall memory space of 1 Gbyte, for a global computation performance of 1 Gflop and an aggregated transfer rate of 640 Mbytes/sec.

5.4. CONCLUSIONS

This chapter has analyzed the class of homogeneous pyramid architectures, covering both compact and distributed implementations. After a look at the common features of the class, a short description of each system has been given.

In regard to compact pyramids, the predominant reason that has prevented the construction of larger systems is the complexity of wiring hierarchical interconnections both at board and system levels. Chip pinout requirements of a pyramid topology are proportional to the number of nodes in the bottom layer embedded in the chip; thus, rather than growing in a linear fashion with the perimeter of the chip, they are roughly proportional to the area. Important advances in packaging technology are required to overcome this problem, so the ratio of core area to periphery area in a chip hosting a subarray of a pyramid structure remains at a good level.

All the implementations of these pyramids have never gone beyond a small prototype. The largest such implementation is a five-level pyramid, while at least 10 levels are necessary to obtain a pyramid capable of tackling real-life image processing problems.

A definite obstacle is, however, fault tolerance. The three-dimensional topology of pyramids prevents using the well-established techniques typical of mesh arrays. No engineered realization of a massively parallel system is ever possible without sound provisions for fault detection and correction. Therefore, while the motivations for compact pyramid systems remain valid, other approaches to the feasibility problem can be explored as discussed in Chapter 7.

The distributed pyramid family, which is based on coarse-grain processing nodes, shares other types of problems. Here the issue is in the arbitration of memory accesses to realize internode communications. The processing node with the largest connectivity ever built has a maximum of eight links, which are not even enough to embed a quad pyramid with 4-connected topology.

Other issues regard the efficient downloading of software from the controlling host and the coordination in SPMD mode of the fully autonomous processing units in the PMMs.

REFERENCES

1 P E Danielsson, Vices and virtues of image parallel machines, in *Digital Image Analysis* (S Levialdi, ed), pp 47–59, Pitman, London (1984)
2 V Cantoni, V Di Gesu, M Ferretti, S Levialdi, R Negrini, and R Stefanelli, The PAPIA system, *J VLSI Signal Process* **2,** 195–217 (1991)
3 S L Tanimoto, T J Ligocki, and R Ling, A prototype pyramid machine for hierarchical cellular logic, in *Parallel Computer Vision* (L Uhr, ed), pp 43–83, Academic Press, Orlando (1987)
4 D H Schaefer, G C Wilcox, and V J Harris, A pyramid of MPP processing elements—experiences and plans, Proc 18th Ann Hawaii Int Conf System Science, Vol 1, 1985, pp 178–184
5 F Devos, A Merigot, and B Zavidovique, Integration d'un processeur cellulaire pour une architecture pyramidale de traitement d'image, *Rev Phys Appl* **20,** 23–27 (1985)
6 M Ferretti, Overlapping in compact pyramids, in *Pyramidal Systems for Computer Vision* (V Cantoni and S Levialdi, eds), pp 247–260, Springer-Verlag, Berlin and Heidelberg (1986)
7 H Li and M Maresca, Connection autonomy in SIMD architectures a VLSI implementation, *J Parallel Distrib Comput* **7,** (2), 302–320 (1989)
8 H Yoda, Y Ohuchi, Y Taniguchi, and M Ejiri, An automatic wafer inspection system using pipelined image processing techniques, *IEEE Trans Pattern Anal Machine Process* **10** (1), 4–16 (1988)
9 W D Hillis, *The Connection Machine,* MIT Press, Cambridge, MA (1982)
10 M J B Duff, Propagation in cellular logic arrays, Proc Workshop on Picture Data Description and Management, 1980, pp 259–262
11 V Cantoni, M Ferretti, and M Savini, Connectivity and spacing checking with fine grained machines, in *Machine Vision for Inspection and Measurement* (H Freeman, ed), pp 85–100, Academic Press, San Diego (1989)
12 H Li and M Maresca, Polymorphic-torus network, *IEEE Trans Comput* **38,** (9) 1345–1351 (1989)
13 D W Blevins, E W Davis, R A Heaton, and J H Reif, BLITZEN a highly integrated massively parallel machine, *J Parallel Distrib Comput* **8,** 150–160 (1990)
14 F A Gerritsen, A comparison of the CLIP4, DAP, and MPP processor array implementations, in *Computing Structures for Image Processing* (J M B Duff, ed), pp 15–30, Academic Press, London (1983)
15 V Cantoni, M Ferretti, S Levialdi, and F Malobertı, A pyramid project using integrated technology, in *Integrated Technology for Parallel Image Processing* (S Levialdi, ed), pp 121–132, Academic Press, London (1985)
16 V Cantoni, M Ferretti, S Levialdi, and R Stefanelli, PAPIA pyramidal architecture for parallel image analysis, Proc 7th Symp Computer Arithmetic, Urbana, IL, 1985, pp 237–242
17 T Fountain, *Processor Arrays Architectures and Applications,* Academic Press, London (1987)

18. G. Gerardi, The PAPIA controller hardware implementation, in *Pyramidal Systems for Computer Vision* (V. Cantoni and S. Levialdi, eds.), pp. 153–164, Springer-Verlag, Berlin (1986).
19. O. Catalano, G. De Gaetano, V. Di Gesù, G. Gerardi, A. Machì, and D. Tegolo, Low level languages for the PAPIA machine, in *Data Analysis in Astronomy II* (V. Di Gesù et al., eds.), Plenum Press, New York (1986).
20. V. Di Gesù An high level language for pyramid architectures in *Pyramidal Systems for Computer Vision* (V. Cantoni and S. Levialdi, eds.), pp. 329–339, Springer-Verlag, Berlin (1986).
21. S. L. Tanimoto, Towards hierarchical cellular logic: design considerations for pyramids machines, Computer Science Dept., Univ. Washington, Technical Report 81-02-01 (1981).
22. S. L. Tanimoto, T. J. Ligocki, and R. Ling, A prototype pyramid machine for hierarchical cellular logic, in *Parallel Computer Vision* (L. Uhr, ed.), pp. 43–83, Academic Press, Orlando, FL (1987).
23. S. L. Tanimoto and J. J. Pfeiffer, Jr., Data processing system having a pyramidal array of processors, U.S. Patent No. 4.622.632 (11/11/1986).
24. S. L. Tanimoto, A hierarchical cellular logic for pyramid computers, *J. Parallel Distrib. Comput.* **1,** 105–132 (1984).
25. J. Serra, *Image Analysis and Mathematical Morphology,* Academic Press, New York (1982).
26. R. P. Blanford and S. L. Tanimoto, A pyramid machine simulator for the symbolics 3600, Proc. IEEE Computer Society Conf. Computer Vision and Pattern Recognition, Miami, FL, 1986, pp. 427–429.
27. D. H. Schaefer, P. Ho, J. Boyd, and C. Vallejos, The GAM pyramid, in *Parallel Computer Vision* (L. Uhr, ed.), pp. 15–42, Academic Press, Orlando, FL (1987).
28. K. E. Batcher, Design of a massively parallel processor, *IEEE Trans. Comput.* **C-29**(9), 836–840 (1980).
29. D. H. Schaefer and P. Ho, Counting on the GAM pyramid, in *Pyramidal Systems for Computer Vision* (V. Cantoni and S. Levialdi, eds.), pp. 125–131, Springer-Verlag, Berlin (1986).
30. A. Mérigot, P. Clermont, J. Mahat, F. Devos, and B. Zavidovique, A pyramidal system for image processing, in *Pyramidal Systems for Computer Vision* (V. Cantoni and S. Levialdi, eds.), pp. 109–124, Springer-Verlag, Berlin and Heidelberg (1986).
31. Y. Ni, A. Mérigot, and F. Devos, Designing a VLSI processing element chip for pyramid computer SPHINX, in *Progress in Image Analysis and Processing* (V. Cantoni *et al.,* eds.), pp. 759–766, World Scientific, Singapore (1990).
32. Y. Ni, Contribution à l'étude des architectures massivement parallèles pyramidales: Réalisation d'un circuit VLSI multiprocesseur et dé definition d'un modèle de contrôle dynamique, Thesis University Paris-Sud, Institut d'Electronique Fondamentale (1990).
33. S. Bouaziz, E. Pissaloux, A. Mérigot, and F. Devos, Some hardward and software considerations for the multi-SIMD control strategy of massively parallel machines, Proc. 5th COMPEURO 91, Bologna, 1991, pp. 180–183.
34. G. Fritsch, Numerical simulation of physical phenomena by parallel computing, in *WOPPLOT 86, Parallel Processing: Logic, Organization, and Technology* (J. D. Becker and I. Eisele, eds.), pp. 40–57, Springer-Verlag, Berlin and Heidelberg (1986).
35. G. Fritsch, General purpose pyramidal architectures, in *Pyramidal Systems for Computer Vision* (V. Cantoni and S. Levialdi, eds.), pp. 42–58, Springer-Verlag, Berlin and Heidelberg (1986).
36. G. Fritsch, Memory-coupled processor arrays for a broad spectrum of applications, in *WOPPLOT 83, Parallel Processing: Logic, Organization, and Technology* (J. D. Becker and I. Eisele, eds.), pp. 158–177, Springer-Verlag, Berlin and Heidelberg (1984).

Chapter 6

Pipeline Multiresolution Systems

As an alternative to the true pyramidal arrays of processors other specialized hardware solutions have been proposed to exploit multiresolution approaches. Among these the most popular is the family of pipeline architectures specialized for decimation and expansion of the image size. The major systems in this class are PVM, and HCL/PIPE. These systems which are designed to perform foveation and tracking processes efficiently, are introduced in some detail.

6.1. INTRODUCTION

As we have seen in Chapter 5 the true pyramid of processors requires very large (and expensive) systems. A solution, which allows the adoption of the strategies for foveation, tracking, and general planning as described in Chapter 2, and which in general cannot achieve the performance of a true pyramid but which has a limited cost, is that based on a pipeline architecture specialized for decimation.

In order to physically implement this approach, the simplest and least expensive hardware paradigm is shown in Figure 6.1. The main pipeline loop includes the image memory that stores all versions of the decimated image, a filter unit which is fed by the subarray element values obtained with suitable delay lines, and a decimator unit which selects a subset of elements that will

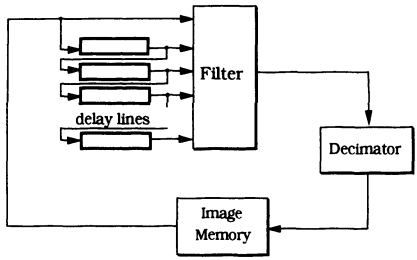


Figure 6.1. The hardware paradigm of a pipeline multiresolution constructor. The data stream on the main loop and recursively the different resolution versions of the image are built, step by step, from the lower level up to the apex (single pixel image).

form the next reduced version of the image. The filter unit may perform convolutions implementing Gaussian and, with additional simple hardware, Laplacian transforms of the original image.

This solution usually includes a dedicated filter which implements the convolution with a limited kernel (in space 3–9 pixels in linear size and with constraints in the coefficients) in one (or few) clock cycle. In this manner, all the multiresolution versions of the image can be obtained in $\frac{4}{3}$ of the scanning time of the original frame. In fact, the scanning of the first level, having only $\frac{1}{4}$ of the image data, requires only $\frac{1}{4}$ of the base scanning time, the second level requires $(\frac{1}{4})^2$, the third $(\frac{1}{4})^3$, and so on, for a total of $\frac{1}{3}$.

A host system, adopting this special hardware, can obtain the multiresolution environment with a delay of 1/3 of the frame time. This multiresolution constructor constitutes the minimum hardware for implementing a "smart sensor" which not only supplies the multiresolution environment but which also, being a convolver, is suitable for implementing low-level operators. Moreover, this specialized coprocessor is usually capable of supplying selected windows of variable sizes and resolutions to the higher-level or host system.

In what follows, the machine conceived almost 10 years ago at the D. Sarnoff Center (Princeton, NJ) by Burt and colleagues,[1] will be described in some detail. This machine has been continuously upgraded with new methodologies and new hardware facilities. Some remarks will be made on the PIPE system,[2] which was conceived to supply the computational framework of the hierarchical cellular logic (HCL) system described in Section 5.2.4.2 at a lower cost (obviously, with lower time performances).

Another system, mentioned in Chapter 2, which can be classified in this family is the GOP (general operator processor)[3, 4] image computer. In fact, even if this system is not conceived as a strictly multiresolution machine, the objective of the designers has been to gain efficiency in the feature pyramid manipulation. The computational paradigm of the system is given in Figure 2.10. The implementation was based on a two-stage pipeline architecture. The

first stage contains the parallel peripheral processor composed of four parallel pipeline units, in which a data stream from the image memory is weighted with kernels extracted from a separate mask memory segment. Content-dependent image filtering can be realized by looping through the feedback structure as shown in Figure 2.10. After the parallel processing phase executed in the first stage, the serial and flexible processing usually done at an higher level of data abstraction is performed on the second stage. On the basis of the described prototype, the proposers developed a new system (GOP 300) which is marketed by Context Vision. This last version includes, besides the hierarchical organization, other special function subunits (in particular, a geometric transform processor and a bit-oriented processor).[5]

6.2. PYRAMID VISION MACHINE SYSTEM

The first pyramid pipeline machine to be designed and built was the pyramid vision machine (PVM). In addition to the construction of the Gaussian and Laplacian pyramids, the PVM was designed to perform fast execution of low- and intermediate-level operators and the foveation process.

The framework of the pipeline pyramid is specialized as shown in Figure 6.2. The purpose of the new blocks (with reference to Figure 6.1) is to allow the implementation of Laplacian pyramids "on the fly."[6]

The Gaussian pyramid **G** is constructed directly by the outer cycle, following Eq. 2.12:

$$G_h = [w * G_{h-1}]_{\downarrow 2} \qquad \forall h, \quad 1 \le h \le n \tag{6.1}$$

having $G_0 = \mathrm{I}$, stored in the image memory, through the expander (set to 1) the data arrive at the filter w (which has the characteristics described in Section

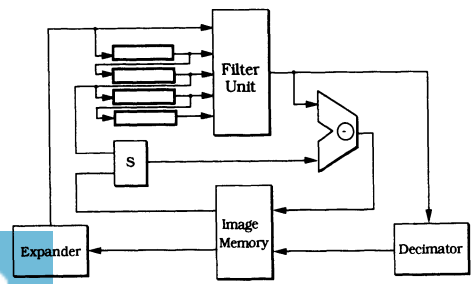


Figure 6.2. A simplified block diagram of the PVM pipeline pyramid processor. One scan of the image sequence is sufficient to implement both **G** and $\mathbf{L}_{FSD}$ while for the complete $\mathbf{L}_{RE}$ representation each step requires two successive scans of the current image representation.

2.4.2) and then go directly to the decimator and back to image memory. By looping onto this cycle, all the Gaussian levels are built into just one scan sequence.

As briefly described in Section 2.4.3, the Laplacian pyramid **L** can be obtained by subtraction of two consecutive Gaussian representations. In order to achieve equal size representations at different levels, the hardware shown in Figure 6.2 supports two solutions. The first solution is called filter–subtract–decimate (FSD) because the successive levels are built by first filtering, then subtracting, and lastly decimating the image. $\mathbf{L}_{\mathrm{FSD}}$ is performed on the fly in just one scan sequence on the basis of the following equation:

$$L_h = G_h - \overline{G}_{h+1} \qquad \forall h, \quad 0 \le h < n \tag{6.2}$$

where $\overline{G}_{h+1}$ represents the partial results, before decimation, in the process of computing the next scaled Gaussian $\overline{G}_{h+1}$. This is achieved on the fly at the ALU output, when the inputs are the filtered and unfiltered G_h (multiplexer S switching the higher input signal). Note that this FSD solution does not allow image recovering from its Laplacian representation: for several applications this is a real disadvantage.

A different implementation that permits the exact image reconstruction is based on the following Laplacian definition, which is called reduce expand (RE)[6] because each Laplacian level is computed after the construction of the successive Gaussian level is accomplished and then reexpanded to the current Laplacian level size:

$$L_h = G_h - G_{h+1,1} \qquad \forall \mathrm{h}, \quad 0 \le h < n \tag{6.3}$$

for $L_n = G_n$ and $I = G_0$, Eq. (2.27) for representation completeness is easily verified. The implementation of Eq. (6.3) requires a second cycle: after having computed G_{h+1}, it is necessary to work with the inner loop, feeding the ALU with G_h (through S, the lower input signal) and carrying the new Gaussian level G_{h+1} to the expander block (for padding) and then to the filter unit (expanding kernel w'). This solution requires a richer hardware, and it can be more time consuming. But **L** achieved with the RE method is a complete transform as required for image coding, transmission, and other applications.

On the basis of the functional diagram of Figure 6.2, a new chip has been designed.[7] The Sarnoff chip has been integrated using 1-μm CMOS standard cell technology in a 300×300 mil die and is packaged in a 84-pin PLCC. It was fabricated by VLSI technology, and the basic clock cycle is 20 MHz. The chip contains a 5×5 filter based on four horizontal line delays of 1024 pixels,

and the frame store is external. The filter can be programmed, with edge control capability for **G**, $\mathbf{L}_{FSD}$, and $\mathbf{L}_{RE}$ pyramid construction, gradient, moment, and subband pyramid transforms and inverse transforms. The filter coefficients can be selected on a limited set (including binomial coefficients), but are rich enough to effectively solve a remarkable range of practical applications. At clock rate, a single chip can compute **G** and $\mathbf{L}_{FSD}$ in less than 18 msec (55 frame/sec) for standard 512×512 images.

This multiresolution constructor can be used as an attached processor, with a low-cost host to solve some specific and limited tasks, or it may be included as a smart sensor in composite systems. In the following sections two cases of integration of the pipeline processor are described: the lattice or segmented pipeline architecture and the CAIP system.

As an attached processor the PVM has been run with several hosts, but most developments have been with an IBM AT.[8] An interesting application of this system (the host in this application was a M68000-based microcomputer),[6] which achieves real-time performance, was the surveillance of a building and its environment in an outdoor scene. In these images the conditions of illumination and the selectivity of the movements to be detected (slow movements like those of clouds should be ignored as well as fast movements of tree leaves due to the wind) further complicate the solution of the task.[9] The solution pursued in this practical case is in principle the same as the one illustrated in Section 2.6.1, particularly the third example, illustrated by the image flow diagram given in Figure 2.11c. Note that this system, which was composed of a few boards, was working before the development of the Sarnoff chip, which can now be adopted to simplify and reduce hardware and cost.

6.2.1. The Segmented Pipeline Architecture

A general vision application requires a task decomposition in which the solution is achieved by a sequence of the available computational primitives; note that the computation time of a single primitive varies as a function of its computational depth and is dependent on the resolution level. For instance, the maximum resolution (which corresponds to the original image) takes the ordinary frame time, while the second level will only require one quarter of that time, etc. The flow diagrams introduced in Section 2.6 illustrate some common sequences of filterlike operations, exploiting multiresolution.

If some buffers divide the sequence into operative blocks having the same resolution or data load (e.g., equal region of interest sizes), each active primitive can run at the maximum design rate. The image data flow through the

processing elements in a pipeline or general data flow modality, where source and destination are the buffer modules.

A schematic representation of the proposed segmented pipeline machine, in which the computation segments can be rapidly allocated to the processing modules in the appropriate sequence, is given in Figure 6.3.[10] It consists of some memory modules (in the first prototype[8] three memory buffers of 512×512 pixels) and some processing modules (in origin, two-cluster modules, one including a multiplier, an ALU, and a shifter, while the second contains the PVM system previously described).

Buffer elements are involved in different activities:

- They are source and destination of the data block of the computation segments; in general, they must have sufficient memory space to store several image data blocks.
- They support data communication between the pipeline machine and the other system units (host computer, MIMD system for high-level processing, etc.). In order to implement these exchanges three mutually exclusive I/O capabilities are adopted: they can serve as input and output data ports within the pipeline machine exchanges and can supply the random data access from outside the machine (shaded lines in Figure 6.3).
- They support decimation, which is executed when data are stored in

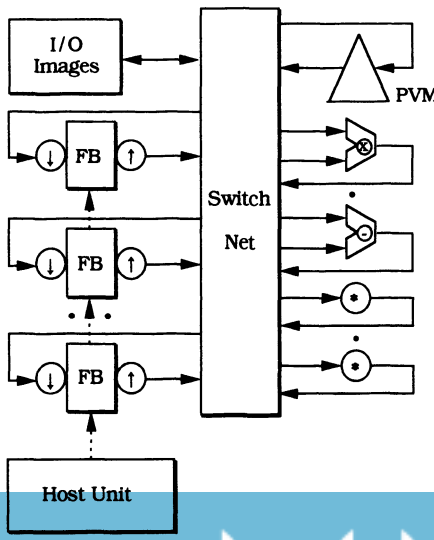


Figure 6.3. The simplified scheme of a segmented pipeline system: (left) image I/O peripherals and frame buffer elements; (right) a collection of different processing operators, including the PVM system for the multiresolution construction environment.

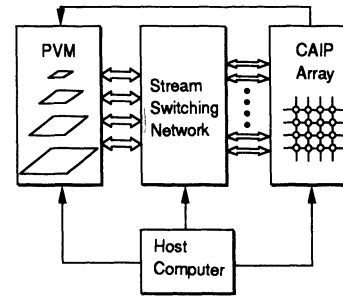


Figure 6.4. The simplified scheme of the CAIP system. The peripheral subunit which implements the multiresolution environment is integrated through a stream switching network with a powerful multiprocessor system.

the buffer, and expansion, when data are transferred from the buffers to the operational units.

- Windowing operations can be performed, even in combination with the expand operation, during data transfer out of the buffers.

The switch network allows flexible interconnection between the buffer modules and the operational units. Multiple sequences of flows from the buffers (even with resampling) to one or more computational units and back to the memory module are allowed simultaneously. The host computer is charged with the control of data flows and computations and of the higher-level vision functions.

The segmented pipeline structure can be extended with a network of such modules realizing a multiple pipeline or *lattice pipeline*[11] in which the branches of the image flow diagram can be executed in parallel and where the flows converge and diverge, accumulating and retransmitting data through the buffers but without loops.

6.2.2. The CAIP System

A conceptually different system proposed around the PVM is the CAIP advanced image processing system (CAIP) conceived at the CAIP Center of Rutgers University.[12] A general scheme of this machine is given in Figure 6.4.

In this machine, the PVM unit works in combination with a multiprocessor system operating in MIMD mode, the CAIP array. Multiresolution is the basic image computational approach of this system. The MIMD unit works in a second hierarchical level dedicated to the (i) analysis of different regions of interest, (ii) investigation of alternative solutions within a region, and (iii) verification of the presence of pertinent features corresponding to a supposed solution.

On this basis the CAIP system architecture is composed of four modules

as shown in Figure 6.4: the pyramidal pipeline unit PVM (able to perform arithmetic and Boolean operations between images, point and local operations like convolution and filtering, gray-level statistics, and decimation in order to implement multiresolution versions of the images), the CAIP MIMD mesh–hypercube array of transputers (in charge of intermediate- and high-level computational stages), and the interface between the two units and the host computer.

The key component of the system is the interface between the smart sensor and the high-level unit. Its main features are the ability to broadcast image segments (regions of interest) from any of the buffers of the PVM to a selected subset of the nodes in the CAIP array; the ''on the fly'' regular distribution of portions of an image to the nodes and, conversely, the collection of data blocks from nodes to eventually form a reconstituted image in the PVM; and image segment exchanges between the processing units of the CAIP array.

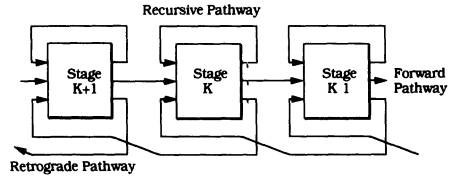
This two-stage vision system, in which low- and high-level computations are performed in two different (integrated) units and which is specialized for peripheral vision activity and intermediate and symbolic processing has been often proposed but never built to its full capability.

6.3. PIPE SYSTEM

A second pipeline processor which supports pyramidal processing is the *p*ipelined *i*mage *p*rocessing *e*ngine (PIPE) developed by the National Bureau of Standards of USA, and University of Washington. As in the HCL pyramid described in Section 5.2.4.2, the plan of the PIPE machine incorporates a set of cellular logic operators (CLOs), based on the quad-pyramid topology, as primitives of the pyramidal algebra of images called *hierarchical cellular logic,* introduced by Tanimoto.[13] Furthermore, the PIPE machine is tailored for the implementation of these operators which constitute a subset of its basic pyramid-mode operations.[14] In fact, each HCL primitive operation corresponds to a single instruction and is executed in a single machine cycle of PIPE.

6.3.1. The PIPE Architecture

The PIPE system is based on a generalized pipeline (shown in Figure 6.5) of a sequence of modular processing stages (MPSs, eight in the prototype). The overall architecture consists of a bidirectional linear cascade of MPSs. Each MPS contains a PIPE processor and two frame buffers and is concerned



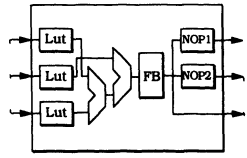


Figure 6.5. The linear sequence of three modular processing stages of the PIPE system Each MPS is connected to two neighbors and can recursively process its own data The combining and distribution logic within the MPS are given in a simplified scheme

with one of the pyramid levels. In the prototype a complete quad pyramid with an image in the base of 256×256 pixels can be contained within the machine (apex in the eighth stage).

The three outputs of each MPS correspond to the three roles of the "home" cell in the pyramidal near-neighbor defined in Figure 5.4. The bottom output corresponds to the father role, so a retrograde pathway leads to the input of the left MPS corresponding to the previous level; the middle output corresponds to the son role, so a forward pathway feeds the successive level (right MPS); finally the upper output corresponds to the intralevel activity with the brothers, and the pathway recursively enters the same MPS.

Images can be transferred from one layer to another in scan mode. When working in pyramid mode, going from the base to the apex, the image size is reduced to one half in each linear dimension by a sampling process which delivers through the forward pathway the odd pixels of the odd rows in a densely packed array. In the opposite direction (apex to base), through the retrograde pathway, data are replicated so that each father is assigned to the four sons.

Obviously, the recursive pathway leaves the image size unchanged. The computation capability of the processor allows each stage to manipulate, through the recursive pathway, a complete $256 \times 256 \times 8$-bit image in $\frac{1}{16}$ sec (image machine cycle). Moreover, the 8 bits/pixel can be manipulated as eight independent Boolean data. In this way 2×8 bit planes can be operated in parallel, reaching the performance of more than 5×10^8 binary operations per second.

6.3.2. Pyramid Neighbor Operations

The basic pyramid near-neighbor operation carried out by PIPE is the CLO matching operation in the forms AND__Match and OR__Match described in Section 5.2.4.2b. The operations involve two operands: the first is the values of the 14 cells of the corresponding pyramidal neighborhood; the second operand P is a pattern which describes the local structuring element. Each of the 14 extended neighborhood values is in ordered correspondence with one digit in the pattern specifying either the 0 or 1 of D (don't care condition) value.

In the AND__Match defined in Eq (5.8), either the result is 1 if each and every element of the pattern matches its associated element in the extended neighborhood, or it is 0 when at least one element does not match. In the OR__Match defined in Eq. (5.9), the result is 1 if at least one element of the pattern, different from D, matches its associated element in the extended neighborhood. It is 0 when no element matches.

The two Match operations are the PIPE pyramidal primitives, and logical and data movement operations must be implemented through them (in PCLIP they were just the primitive, but here the instruction set is much wider).

The bits of the 3×3 subarray (e.g., home cell and brothers) in PIPE are packed into a 9-bit vector which addresses a 512-entry look-up table (LUT). If the LUT has previously been loaded with the correct result of the requested matching with P, the intralevel AND__Match or OR__Match operations can be executed in one machine cycle (in fact, the hardware supports the parallel execution of two independent operations of this kind). In the same machine cycle this result can be delivered in parallel to the connected MPS.

This capability allows us to execute the complete AND__Match or OR__Match operation over the 14 pyramid neighbors in a two-step cycle. In the first step the NOP-1 and NOP-2 operational blocks in Figure 6.5 execute the matching between the intralevel home brothers and sons, respectively (AND__Match and OR__Match operations are decomposable and commutative). The three MPS outputs deliver, respectively, the father (retrograde pathway), the partial matching of the sons (forward pathway), and the partial local matching of home and brothers (recursive pathway). All stages receive the partial results in the three inputs (the father data are matched in the entry LUT of the retrograde pathway). The cascade of the two ALUs congruently accomplishes the matching, in the order of father and home, and finally matches this result with the son (see Figure 6.5). Note that this operation can be executed in parallel for eight bit planes.[14]

The AND__Match or OR__Match operations can be executed iteratively and even recursively until no further change occurs (see Section 5.2.2.1b).

This control is based on a UNTIL NO CHANGE metaoperator that occupies at the most a machine cycle for some HCL operations (in some cases it can be incorporated into the iterations).

6.4. CONCLUSIONS

This chapter has analyzed the class of pipeline pyramid architectures, which usually represents a much cheaper alternative to the true hardware pyramidal systems described in Chapter 5. After a look at the common features of the class, a short description of the two main systems has been given.

The basic framework of this class is to compose a two-level system. The first subsystem acts like the visual periphery in humans, applying simple operators to a large amount of data, thus forwarding to the higher subsystem the condensed information extracted. The higher subsystem drives the former subsystem, which is often considered a "smart sensor," to obtain additional investigation (in the human analogy this is the role of the inner cortical areas).

Presently, very interesting systems are coming out, often many orders of magnitude cheaper than the massively parallel systems. Nevertheless, the weak points of this framework still remain the lack of flexibility in the low-level stage and the crucial integration between the pictorial processing capability with the symbolic capacity of the host or high-level subsystem.

REFERENCES

1 P J Burt and G van der Wal, An architecture for multi-resolution, focal, image analysis, Proc 10th Int Conf Pattern Recognition, Atlantic City, NJ, 1990, pp 305–311
2 E W Kent, M O Shneider, and R Lumia, PIPE pipelined image processing engine, *J Parallel Distrib Comput* **2**, 50–78 (1985)
3 G Grandlund and J Arvidsson, The GOP image computer, in *Fundamentals in Computer Vision* (O Faugeras, ed), pp 443–458, Cambridge University Press, Cambridge (1983)
4 K Lundgren, D Antonsson, J Arvidsson, and G H Grandlund, GOP, a two stages microprogrammable pipelined image processor, Proc 2nd Scandinavian Conf Image Analysis, Helsinki, Finland, 1981, pp 408–414
5 G Grandlund and J Arvidsson, Computer architectures for image processing, Proc 4nd Scandinavian Conf Image Analysis, Trondheim, Norway, 1985
6 P J Burt, C H Anderson, J O Sinniger, and G van der Wal, A pipeline pyramid machine, in *Pyramidal Systems for Computer Vision* (V Cantoni and S Levialdi, eds), pp 133–152, Springer-Verlag, Berlin (1986)
7 G S van der Wal, The Sarnoff pyramid chip, Proc Workshop on Computer Architectures for Machine Perception (B Zavidovique and P L Wendel, eds), Paris, 1992, pp 69–79

8. P. J. Burt, Smart sensing in a pyramid vision machine, *Proc. IEEE*, **76**(8), 1006–1014 (1988).
9. C. H. Anderson, P. J. Burt, and G. van der Wal, Change detection and tracking using pyramid transform techniques, Proc. SPIE Conf. Intelligent Robots and Computer Vision, 1985, pp. 72–78.
10. P. J. Burt and G. van der Wal, An architecture for multiresolution, focal, image analysis, Proc. 10th Int. Conf. Pattern Recognition, Atlantic City, NJ, 1990, pp. 305–311.
11. P. J. Burt and G. van der Wal, Iconic image analysis with the pyramid vision machine, Proc. Workshop on Computer Architecture for Pattern Analysis and Machine Intelligence, Seattle, WA, 1987, pp. 137–144.
12. V. Cantoni, K. N. Matthews, P. Burt, H. Freeman, A. Terrano, and G. van der Wal, CAIPS—the CAIP advanced image processing system, CAIP-TR-072, pp. 1–39, CAIP Center, Rutgers University, 1988.
13. S. L. Tanimoto, A hierarchical cellular logic for pyramid computers, *J. Parallel Distrib. Comput.* **1,** 105–132 (1984).
14. E. W. Kent and S. L. Tanimoto, Hierarchical cellular logic and the PIPE processor: structural and functional correspondence, Proc. IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Image Database, 1985, pp. 311–319.

Chapter 7

Simulation of Pyramids on Flat Arrays and Hypercubes

This chapter reviews different solutions to embed pyramidal processing in massively parallel systems, such as mesh arrays and hypercubes. The first approach of using a standard mesh array computer to obtain multiresolution processing has a particularly high simulation cost for interlevel data exchanges given the simpler paths available in the mesh. Increasing the connectivity of mesh arrays by the addition of a few links per PE opens up a number of alternatives. These are analyzed in terms of regularity of decomposition and fault tolerance. However, the appropriate use of a switch lattice which interconnects standard PEs of a mesh array allows us to embed quad pyramids into a highly regular physical structure.

The hypercube topology can be used in two situations. Either embedding the pyramid as a set of meshes or directly mapping the nodes without explicitly reproducing the near-neighbor connectivity within the levels. The former approach allows interlevel communication by using the properties of reflective Gray codes. The latter reduces intralevel communication cost, but gives rise to rather irregular communication patterns in the hypercube dimensions.

7.1. INTRODUCTION

As indicated in previous chapters, pyramid machines already built are still at a prototype stage. It is therefore no surprise that applications in the real

world cannot be run reliably on them. There have therefore been many efforts to use massively parallel systems with other topologies as if they were pyramids. Since commercially available massively parallel systems fall within the three topologies of linear, mesh array, and hypercube systems, most work has been concentrated on them, especially the last two. Indeed, the linear topology seems too far removed from the bidimensional support on which a pyramid is built.

Indeed, the problem of transferring a set of parallel algorithms, designed to run on a pyramid computer, onto a different parallel architecture is an instance of the more general problem of mapping a regular architecture A_1 onto another regular architecture A_2. Such a problem, which has been investigated in many single combinations such as tree embedding in meshes, tree embedding in hypercubes, and so on, has also been addressed in its widest formulation by Stout.[1] He considers all possible combinations among the most-used architectures, along with lower and upper bounds in algorithm complexity. The main reason that motivates such an approach is not so much efficiency as it is the correctness of the algorithms. Indeed, while the mapping of A_1 onto A_2 generally introduces large overhead, it does guarantee that an algorithm known to be error proof in A_1 will be error proof in A_2.

There are two aspects that influence efficiency: the mapping itself, which must be as good as possible to lower any overhead and the structure of the algorithm. That being designed for the source architecture A_1 is likely to be far from the best in the A_2. On average, the best possible algorithm for A_1 will be considerably less efficient than the best one for A_2, and usually the loss due to mapping will be large.

Common parameters used to measure the efficiency of the mapping are

- *Processor load.* The maximum number of processors of A_1 simulated by a single processor in A_2 (in some cases, the notion of ''virtual processor ratio'' is also used for this purpose). This measures both the computation congestion and the memory congestion.
- *Expansion.* The ratio between the number of processors in A_2 and that in A_1. This parameter is not influenced by the mapping strategy, but if the structures are modular and mapped with a balanced distribution, the inverse of the processor load results come close to this parameter.
- *Link load.* The maximum number of communication links in A_1 mapped onto a single communication link in A_2. This is a measure of traffic congestion.
- *Dilation.* The maximum number of communication links in A_2 necessary to map a single communication link in A_1. This is a measure of

the maximum delay for simulating data exchanges between a pair of pyramid nodes (PNs).

While this mapping approach sets a formal basis to the simulation problem, which is essentially a software task, other considerations motivate a deeper insight into the architectural issue. There exist proposals to build reconfigurable parallel architectures[2] capable of embedding a few alternative topologies by properly choosing connectivity links by means of software. This is possibly a viable solution to the mapping problem as well. Furthermore, a few proposals to increase basic topologies, namely meshes, with extra links on top of those required by 4- or 8-connectivity are also worth considering.

The remainder of the chapter is split into two sections. The first covers targeted architectures based on the mesh topology and analyzes both pure software simulations and enhanced–reconfigured systems working as pyramids. The second addresses the hypercube family, for which only simulations have been proposed.

7.2. PYRAMIDS AND MESHES

The pyramid and mesh topologies are clearly closely related; a pyramid can be seen as a stack of meshes of edge length decreasing with level. Most naturally, the mapping of a pyramid architecture onto a mesh computer starts by first establishing a correspondence between a single level of the pyramid and the available mesh system, according to the size of the last one. The ways in which the resources of the mesh are used to establish this correspondence and the others necessary for the remaining levels are addressed as a software simulation problem. Increased meshes and reconfigurable meshes, on the other hand, try to obtain an emulation, even if partial, of the pyramid. This is achieved by using particular hardware features, notably communication links, as subsets of the true pyramid.

7.2.1. Simulation of Pyramids on Flat Arrays

There are two approaches to the simulation of the pyramid on a flat array. The first is based on the concept of distributing the data across the mesh of PEs in order to preserve adjacency and locality among PNs; the second is based on distributing the PEs across the data. In what follows, an instance of the mapping for each approach is described with a quantitative assessment of performances.

7.2.1.1 Compact Simulation

The first proposal of pyramid simulation on flat arrays has been described in Reeves.[3] This strategy depends heavily on the efficiency of mapping interlayer communications on the simpler paths available in the mesh.

In order to map a pyramidal structure which has a $K \times K$ base on a flat array of dimensions $N \times N$, a suitable allocation of the PNs has to be performed according to the level to which they belong. A PE of the mesh can carry out the functions of a set of PNs, which are mapped in its own local memory. In general, PNs belonging to levels whose dimensions are less than $N \times N$ are mapped into a single memory layer. PNs which belong to the level of $N \times N$ dimension are mapped into a second memory layer, and PNs of the lower levels are mapped in several layers, as explained later. If one considers a k-level pyramid ($k = \log_2 K + 1$), where $n = \log_2 N$, three cases can be considered: (i) $k \leq n$, all the pyramid levels are allocated in one memory layer with some idle PEs; (ii) $k = n + 1$, two memory layers are necessary, one for the pyramid base only and a second for the remaining $k - 1$ levels; (iii) $k > n + 1$, the n upper levels are allocated in just one memory layer, while the remaining $k - n$ levels are allocated in successive memory layers, whose total number M is equal to

$$M = \tfrac{1}{3}(2^{2(k-n)} - 1) \tag{7.1}$$

For example, if a 16×16 pyramid ($k = 5$) is simulated on a flat array of 8×8 processors ($N = 8$, $n = 3$), the three topmost levels (from the apex) are mapped into the same memory layer, while levels 3 and 4 (for a total of 320 PNs) are mapped in five successive memory layers, according to Eq. (7.1). Figure 7.1 shows an example of the mapping of such a pyramid on an 8×8 mesh.

This method has considerable problems in terms of memory requirements of the PEs as the number of levels of the pyramid increases. Moreover, the load of the PEs in simulating the PNs is unevenly distributed in the mesh.

Data movements are performed in different ways according to the level of the pyramid in which they take place. For the topmost n levels, vertical communications toward lower levels are implemented by a bidimensional perfect shuffle and by a reverse shuffle in the opposite direction. For horizontal data movements, the shift technique is adopted.

For what concerns the other k-n levels, two mapping strategies can be followed: the former has been called the *crinkled storage format*[3] and stores a complete subpyramid of k-n levels in each PE. The latter is based on mosaicking lower levels of the pyramid by square tessels of dimensions which match

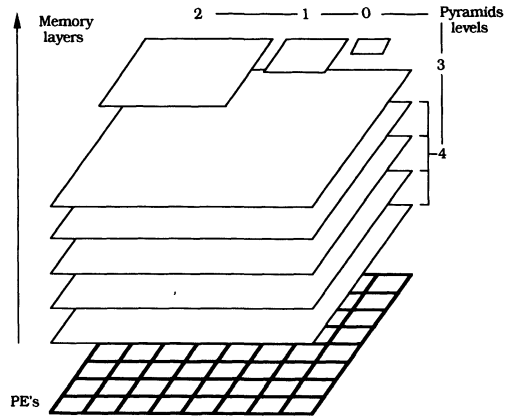


Figure 7.1. Embedding of a five-level quad pyramid into an 8×8 mesh. Note that the pyramid base is mapped on $2^{2(k-n-1)}$ memory layers of the mesh.

the mesh array. The former solution allows local operation for vertical and horizontal exchanges inside the subpyramids, but prevents using the parallel near-neighbor access. The latter, however, maintains this parallel access capability partially, but to the detriment of the link load, which increases roughly by a factor of M. For this reason we will subsequently only refer to the crinkled solution.

In crinkled mapping, the data shifts are not useful for interlevel exchanges. This is because data communication in these cases is performed by local accesses in the memory of the PEs. Instead, exchanges among siblings are local for horizontal communications, while general near-neighbor access involves inter-PE communications.

In the compact simulation, for the higher levels, vertical interchanges are not implemented in an efficient way. This is due to the shuffle operations overhead, whose complexity is $O(N)$. On the other hand, hardware topology is well exploited in intraplane communications, thanks to lateral near-neighbor connectivity. Vice versa, vertical communications are relatively convenient, for the base and the lower levels, but horizontal communications cannot benefit from the broadcasting–gating solution.

In Reeves,[3] the time efficiency of the simulation of a pyramid on a flat array is estimated as a function of the ratio between the pyramid base width and the flat array size. The comparison with a true hardware pyramid outlines the fact that pyramidal algorithms can be implemented on a flat array with satisfactory results in particular cases, such as multiresolution operations (e.g., pyramid construction) if integer or real data are used. This is because the shuf-

fle operations can be implemented by pipelined data transmissions, and the resulting overhead is then distributed over a number of bits. All in all, the pyramidal structure is superior to the flat one, especially if the computation is essentially based on hierarchical Boolean near-neighborhood processes.

Considering the four performance parameters introduced in Section 7.1, closed-form formulas for the mapping of a complete pyramid have been subsequently given. They are evaluated for three cases of 8- and 10-level quad pyramids simulated by a flat array of 128×128 PEs (such as the MPP) and for a reduced pyramid implementation consisting of only three levels (from the base to the level which matches the mesh array)

- *Processor load* is $1+M$ [see Eq. (7.1)], which corresponds to 2, 22, 21, respectively, in the case studies.
- *Expansion* is $3N^2/(4K^2-1)$ and $3N^4/(4K^2N^2-4K^2)$ for the complete and truncated cases, respectively. These correspond in the case studies to approximately 0.75 and 0.0468 in the last two cases (and the inverse ratio is respectively 1.33, 21.33). In these cases it is quite close to the inverse of the processor load, given that the simulation load is almost balanced on the array (obviously the higher the processor load value the closer it is to the inverse of the expansion parameter).
- *Link load* is $2^{k-n}+2^{n+1}$ at the common edge between planes $n-1$ and $n-2$, in which horizontal communications of the subpyramid (k-n levels, see above) are superimposed on the vertical communications between planes n with $n-1$ and $n-1$ with $n-2$. In the case studies this corresponds to a load of 258, 264, and 7, respectively (in the first two cases 256 is due to the shuffle operation used to implement vertical communications in the upper levels).
- *Dilation* is N for vertical communications between planes n with $n-1$. Indeed the shuffle operation requires N near-neighbor exchanges in this case (half in each direction). In the first two case studies the dilation is 128, while for the truncated pyramid the value is 1.

7.2.1.2. Distributed Simulation

A second mapping of a quad pyramid into a mesh architecture is obtained with a three-step procedure as follows[4]:

1. by flattening the three-dimensional topology of the pyramid (a nonplanar graph) into the planar graph embedding the base of the pyramid;
2. by slicing the degenerate pyramid thus obtained into submeshes, each matching the target mesh architecture;
3. by stacking each such submesh onto the target mesh without folding or rotations.

The resulting solution is quite different from the compact pyramid simulation case. The distributed solution loses adjacency in levels above the base. The exchanges between these layers improve partially with the distribution of higher-level nodes.

The first step is a recursive procedure which starts from the planar graph embedding the base of the pyramid. Within each 2×2 block of sibling nodes, one node is chosen to act as the parent. Its link with the parent node is broken while the remaining three are flattened to match the links between siblings. The process of contracting the graph terminates when the apex of the pyramid is collapsed into the corner node of the base graph. There are four possible final mappings, selected on the basis of which node acts as the parent within each 2×2 block.

Figure 7.2 shows the mapping of a four-level quad pyramid into an 8×8 mesh array. The pyramid graph has been flattened recursively toward the top left-hand corner. In this picture each processor of the mesh is shown as a number, indicating the maximum level it belongs to (note that the levels of the pyramid are numbered from the vertex downward to the base, which has the label 0). In addition to this level, a single processor belongs also to all inferior levels (e.g., levels 0 and 1 for a processor labeled 2). This solution distributes the PEs over the pyramid graph and places the highest computational load in the single PE belonging to all levels; the quantitative assessment through the chosen descriptive parameters reflects this situation.

Data movements among logically neighboring nodes are time consuming

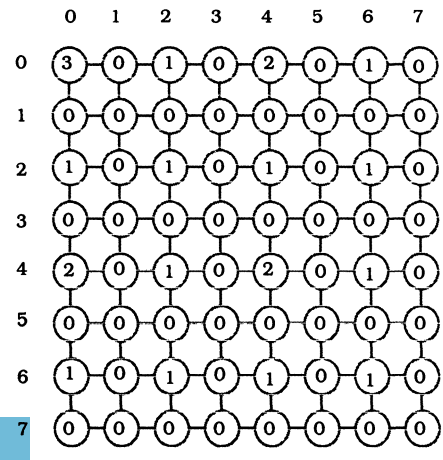


Figure 7.2. Embedding of a four-level pyramid on an 8×8 mesh array obtained by distributing the PNs. The labels represent the maximum levels that the PEs are working for, with the numbering scheme going from 0 (base) to 3 (apex).

for horizontal and, even relatively, for vertical exchanges. Indeed, physical contiguity of PEs is optimal only for lateral exchanges in the base of the pyramid, when it matches the dimensions of the mesh ($k=n+1$) or when it is smaller than this dimension ($k<n+1$). In this latter case, at most only one quarter of the mesh is used. In these cases, neighboring nodes in the upper levels are located on PEs at a distance equal to 2^h-1, h being the label of the level, according to the numbering scheme just introduced. The situation is just a little worse than average, when the pyramid base does not fit completely within the mesh. The k-n topmost levels of the pyramid are all mapped in the single corner PE and thus benefit both from vertical and from horizontal exchanges, which become local memory accesses. Moreover, each PE stores the nodes of the ''home'' quadrant for all remaining levels, as well as the nodes located in the same level at a distance multiple of 2^n for each coordinate. Horizontal communications therefore have a maximum cost in these levels, equal to 2^n-1. Vertical ones have a maximum cost of 2^{2n-2}, which is the cost to send data from the level-n nodes (all mapped in the same corner PE) to the child located on the diagonal direction.

The quantitative analysis is carried out taking into consideration the same cases of 8- and 10-level quad pyramids respectively (in this second case also with a truncated solution). These are simulated by a flat array of 128×128 PEs, as has been done for the compact pyramid analysis.

- *Processor load* is $(n+\frac{4}{3})\ 4^{k-n-1}-\frac{1}{3}$, which corresponds to 8 and 133, respectively, for the complete pyramid case studies, while this last case is reduced to 63 for the simulation of the first three levels (from 512×512 to 128×128).
- *Expansion* as for the previous simulation strategy is $3N^2/(4K^2-1)$ and $3N^4/(4K^2N^2-4K^2)$ for the complete and truncated cases, respectively. These correspond in the case studies to approximately 0.75 and 0.0468 (and the inverse ratio is respectively 1.33, 21.33). In all cases it is too far from the inverse of the processor load, given that the simulation load is concentrated in the vertex location (or in the PEs of highest level).
- *Link load* for the complete pyramid is $2n\ 4^{k-n-1}$ in the vertex corner in which horizontal communications of the lowest n levels are superimposed onto the vertical communications between the same planes (obviously the communications between nodes of the highest subpyramid of k-n levels are solved in local memory). In the truncated simulation the maximums are localized between the highest-level PEs and their near neighbors. For the case studies the values are, respectively, 14 and 224 in the complete case and 147 in the truncated solution.

• *Dilation* is N in the complete cases, as for the compact simulation, for vertical communications between planes n and $n-1$, in which N near-neighbor exchanges are required. For both case studies, dilation is 128. However, for the truncated solution, this parameter is reduced to 4!

7.2.2. Augmented Flat Arrays

The quantitative analysis carried out on the simulation of pyramid on a mesh architecture has highlighted different sources of overhead. According to the intended main use of pyramid computations, either processor load or link load can become a serious bottleneck in using the mesh and should be reduced. We will look at a few proposals for enhancing the basic structure of the mesh with minimal hardware additions, aimed at lowering either parameter.

A method has been proposed by Duff[5] for obtaining a more effective resolution reduction in a flat array by adding an extra connection to a subset of the PEs of the array. This proposal tries to reduce both link load and dilation. However, it does not improve processor load.

In a flat array with 4-connectivity the reduction process can be accomplished by the following means: one parallel shift in the horizontal direction, plus an associative operation and a parallel shift in the vertical direction and a further associative operation. By these means the pyramid data structure is built on top of the base array. However, the resulting pixels will now be distributed across the array and will need to be compacted into a contiguous array in order to reestablish local neighbor connectivity (reverse shuffle). This can be achieved by means of a series of masking and shift operations. This is by far too slow to perform in an array with conventional nearest-neighbor connectivity.

As a possible solution, it has been suggested to add a connection to the processor in the array position (p, q) to the processor with coordinates $(2p, 2q)$, for all integer values of p and q (see Figure 7.3a). These extra connections allow us to concentrate data with just one parallel shift.

Although it is clear that this solution can be an efficient means of performing typical pyramidal operations in a flat array, some critical issues need to be pointed out. A first objection regards the irregular structure of the resulting array, with connections of different lengths, introducing a specialization of the processors and resulting integration problems. In Figure 7.3b an improvement is shown. In this solution the horizontal and vertical regularities are gained at the expense of an increase of the dilation parameter. Every processor with coordinates (p, q) is connected to the processors in the array position $(2p,$

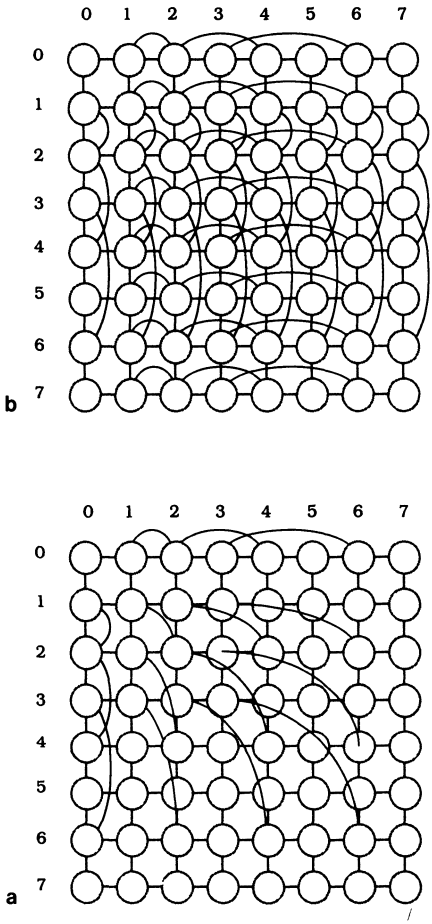


Figure 7.3. Embedding of a four-level pyramid on an 8×8 mesh array obtained by compacting the PNs. In (a) just one link is introduced in order to reduce link load and dilation. In (b) a more regular structure is obtained by breaking the extra connection into horizontal and vertical paths (a maximum of two extra links per node).

q) and (p, $2q$); in this way the unit path of the solution shown in Figure 7.3a is split into horizontal and vertical components. As can be seen, all rows (columns) are equal concerning the connections along each row (column).

Moreover, every processor of a given level also belongs to all inferior levels, placing on it the burden of larger quantities of data as the level increases. This is especially true for the vertex of the pyramid, corresponding to the processor in the top left corner. A third problem, also derived from the specialization of the processors, is that of fault tolerance, which seems to be very poor in this architecture, due to the fixed structure of the interconnections.

However, both these solutions do not show a satisfying degree of modularity, due to pinout problems. Indeed, for a chip containing $P \times P$ processors, $O(P^2)$ external connections must be provided.

The usual quantitative analysis is carried out considering the same cases of 8- and 10-level quad pyramids simulated by an augmented flat array of 128×128 PEs, as for the compact pyramid analysis. In particular, the crinkled solution is considered (since no advantage is achieved because the mosaicking operation does not overlap the extra paths). Moreover, no reports about the truncated case are given. Indeed, for this case the solution coincides with the given compact simulation. Both the (a) and (b) solutions are considered, even if they have the same performance for the processor load, expansion, and link load parameters:

- *Processor load* is $n+M$, which corresponds to 8 and 28, respectively, for the complete pyramid case studies.
- *Expansion*, as for the previous simulation strategy, is $3N^2/(4K^2-1)$ corresponding for the case studies to 0.75 and 0.0468 (and the inverse ratio is respectively 1.33, 21.33).
- *Link load* is $3n-2+2^{k-n}$ in the vertex corner in which horizontal communications of the n higher levels are superimposed onto the $2n$ vertical communications between the same levels (obviously all the communications between nodes of the lowest subpyramid of k-n levels are solved in local memory) and onto the lateral communications of the k-n subpyramid. For the case studies the values are, respectively, 21 and 27.
- *Dilation* in each p,q node is equal to 3 and 4 for the (a) and (b) solutions, respectively.

7.2.3. Reconfigurable Meshes Emulating Pyramids

Pyramids have also been simulated with reconfigurable massively parallel meshes. In these cases no extra links have been introduced, and exchanges between distant PEs exploit the reconfiguration capability of the array. Two proposals that distribute the pyramid PNs spatially and functionally in reconfigurable meshes will be described. In the first, the PE array is mixed with a switch lattice which creates a logical neighborhood out of PEs physically distributed in the mesh. This first solution has been called *the flat pyramidal architecture*.[6, 7] The second solution, proposed by Maresca and Li,[8, 9] exploits a restricted local autonomy of PEs in setting up near-neighbor connections. In the first case the PNs mapping is static for all levels above the base. In the

second case the role of the PEs changes dynamically during the execution, setting up one level at a time.

7.2.3.1. The Flat Pyramidal Architecture

The embedding of a four-level quad pyramid in a cellular array of 8×8 PEs is shown in Figure 7.4, where each label (✛ , 1, 2, 3 or 4) symbolizes a PE. It is a regular and recursive structure and is easily enlarged. That is, the apex of a five level pyramid (labeled 4) will be located in the bottom-right position of Figure 7.4, while the array is replicated four times.

Each PE in the array performs the functions of at the most two PNs, each belonging to a different plane (grids), the base plane (level 0) and a second, higher-level, one. In Figure 7.4, numeric labels specify the higher level, while the symbol ✛ identifies nodes belonging to the base only. PEs working on two levels are $(N^2-1)/3$. In the following, let us indicate this set of PEs as Y and the complete set of PEs as X, while W will be the set $X-Y$.

If (0, 0) are the coordinates of the top-left element in the array, a PE situated in position (i, j) belongs to the set Y if and only if in the binary representation of both $i+1$ and $j+1$ the least significant nonzero bits occupy the same position h (beginning the count from the rightmost bit). In this case, h is the level to which the PE belongs. For example, the PE with coordinates (3, 6) (then $i+1$, $j+1$; in binary codes, . . . 0$\underline{1}$00 and . . . 011$\underline{1}$) will be a member of the set W because the positions of the least significant nonzero bits are unmatched. The PE in (5, 1), however, will belong to level 2, since the least significant nonzero bits in both the incremented binary codes . . . 01$\underline{1}$0 and . . . 00$\underline{1}$0 are located in the second position.

Conversely, the coordinates in the mesh array of the PN (h, i, j), which belongs to level h with $1 \mathrm{LE} h \leq n$, are given by the expressions

$$[(2i+1)2^{h-1}-1,\ (2j+1)2^{h-1}-1] \quad \forall i, j, \quad 0 \leq i, j < 2^{n-h} \tag{7.2}$$

Furthermore, the generic PE of the set Y with position (i, j) is the apex of a subpyramid whose nodes are mapped to PEs with coordinates (p, q), where

$$i-d \leq p \leq i+d, \quad j-d \leq q \leq j+d, \qquad d=2^{(h-1)}-1 \tag{7.3}$$

Regarding connections, each element of the mesh array (that are all supposed to be identical) is connected with its eight neighbors by a structure

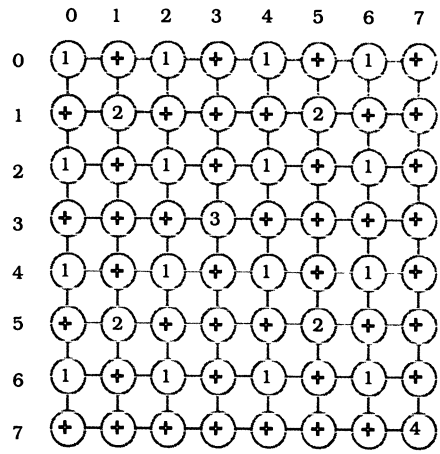


Figure 7.4. Implementation of a four-level virtual pyramid on an 8×8 mesh array featuring 8-connectivity. The numerical labels identify the subset of PEs working on two different levels. PEs labeled with ✛ preserve the parallel access capability to near neighbors for PNs of all higher levels. Note that to simplify the drawing only the 4-connectivity of the horizontal modality is shown.

(switch lattice) that will be described later. This 8-connectivity can be directly exploited only for the base of the pyramid, but is needed for interplane communications.

A basic feature in some fine-grained machines is the parallel access under 4-connectivity to the nearest neighbors of each element. This is preserved on all planes of the logical pyramid. If we consider the generic plane $h > 0$ (whose PEs belong to the set Y), the physical distance between two neighbors is 2^h (see Figure 7.4). Adjacency, however, may be saved, implementing a short circuit between ports east–west and north–south in the PEs of the set W. This situation is depicted in Figure 7.4, where the short-circuited elements (PEs of set W) are properly represented by the symbol ✛. As can be observed, horizontal interconnections on different levels ($h > 0$) exploit disjointed paths.

When operating on the base, at maximum resolution, only a fully SIMD operational mode can be used on the flat array with 8-connectivity. Otherwise, limiting all higher levels to 4-connectivity, given that the grids are disjoint, a Multi-SIMD operational mode is introduced into the pyramid (each grid above the base can operate autonomously in SIMD mode).

Regarding ''vertical'' interprocessor communications, three cases can be identified.

1. Communications between parents of level 1 and children of level 0 (base) are immediate and exploit 8-connectivity in SIMD mode. Each PE of level 1 communicates with three children, situated to the east, to the southeast, and to the south, while the fourth child is simulated by the PE itself. For example, in the top left 2×2 subarray of Figure 7.4,

1 ✢
✢ 2

the four elements are the children of the PE labeled 1.

2. Vertical data exchanges between levels 1 and 2 exploit diagonal links from northeast to southwest, and vice versa, and from northwest to southeast, and vice versa.

3. Communications between levels h and $h+1$, with $h \geq 2$, are finally accomplished by short-circuiting PEs belonging to level $h-1$ and lower levels down to level 1 along the northeast to southwest and southeast to northwest directions. In Figure 7.4, for instance, the third level element in position (3, 3) can send data to its northwestern child with coordinates (1, 1) through the element labeled 1, situated in (2, 2). At the same time, the PE (3, 3) can also broadcast to its other children (1, 5), (5, 5), and (5, 1) in the same way.

No further links are required between nonadjacent PEs. We stress the high simplification obtained by this logical pyramid in inter- and intrachip and board interconnections, in comparison with other physical or simulated pyramidal solutions. Lastly, note that fault tolerance mechanisms are also simplified and will be described in detail later.

a. Some Notes about Reconfiguration. The proposed solution can be considered as a partially reconfigurable SIMD fine-grained array. Reconfiguration is achieved through a limited implementation of connection autonomy.[10] Indeed, the interconnectivity of the flat array described here gives rise to three architectures out of an $N \times N$ physical arrangement of PEs:

1.	A single 8-connected mesh	$N \times N$ PEs
2.	n 4-connected meshes	$2^{n-h} \times 2^{n-h}$ PEs with $1 \leq h \leq n$
3.	A quad tree	$1+n$ levels

Note that in the third case at most two adjacent levels of the tree can exchange data at the same time. Each configuration is mutually exclusive of the other two. The pattern required to build each configuration can be chosen from among three prewired modes that are available at each node of the physical structure. Actual settings depend on the intended logical position of the PE within the structure [that is, on its (i, j) coordinates] but cannot be altered in an arbitrary way.

There is, however, a second, quite different form of reconfiguration available in this system. Since the role played by each PE can be changed, although in limited ways, the embedding of the 4-array tree is completely arbitrary. Any of the PEs can be selected to act as the root of the tree. In other words, the apex of the pyramid can be relocated anywhere on the physical array. Thus, it

is possible to focus on an area of the image wherever interest operators and planning strategies so require. The pyramid is "virtual," and its embedding in the underlying physical mesh array is not committed to defined locations but can be varied with considerable freedom.

In order to maintain the near-neighbor parallel access capability, the mapping of nodes onto PEs must follow a block-partitioning distribution whenever the virtual pyramid is larger than the physical array (i.e., in this case the crinkled format as described in Section 7.2.1.1 is not used, but the image-to-PE distribution is adopted). In detail, either mosaicking or folding can be used (stacking the blocks onto each other in the former case directly, in the latter by row and/or column transposition). The actual solution depends on the available hardware connectivity on the array borders. While folding is always possible, mosaicking is more convenient when the array has a wrap-around capability. Furthermore, nodes belonging to the topmost section of the pyramid, which are mapped in the bottom-right corner PE, can be redistributed following the virtual strategy in the middle of the array.

b. Implementation Notes. According to the criteria so far illustrated, the project of a real machine has been defined under the name of PAPIA 2^7 by a group of Italian researchers. The need to reconfigure the PE array according to different operating modes has motivated the introduction of a switch lattice—a regular structure consisting of reconfigurable switches driven by particular control lines. These switches are regularly connected to the grid of PEs, which are not directly linked.

Each reconfigurable switch may be characterized by four parameters:[11]

1. The number m of lines entering the switch from each direction (data path width)
2. The total number d of paths which enter and exit from the switch (degree)
3. The number c of possible configurations for the switch (configuration settings)
4. The number g of different paths that the switch can simultaneously connect (crossover capability)

The adopted switch-lattice solution, which reduces the number of physical connections of each PE (and consequently of each chip), is illustrated in Figure 7.5. All switches occupy a central position amid four PEs, each connected with the switch by two one-directional links. The representative parameters of the switches used in this solution are $m=1$, $d=8$, $c=3$, $g=4$.

The three configuration settings are detailed in Figure 7.5a, b, and c. The

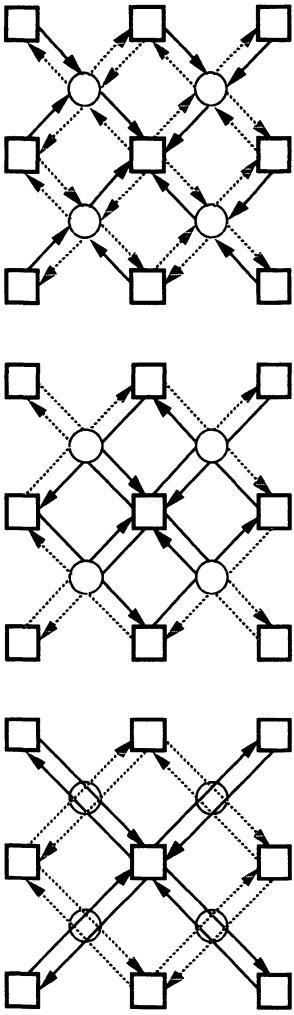


Figure 7.5. Layout of PEs and switches distribution in PAPIA2 (PEs are represented by squares, the switches by circles): (a) the active links for the central PE of a generic 3×3 subarray are detailed in the case of base operations; (b) setup of the switches for the horizontal operations; (c) switch setup for vertical operations.

first is exploited when operating on the base of the pyramid. Note that, 8-connectivity is achieved by partially distributing the gate logic described in Section 5.2.2.1 among the switches. The second setting allows horizontal connections among PEs belonging to levels above the base (with 4-connectivity). The third accomplishes vertical connections between couples of adjacent planes, except planes 0–1.

c. Fault Tolerance in PAPIA 2. Among the alternative strategies for achieving fault tolerance in fine-grained mesh arrays,[12] a simple, yet popular, strategy for arrays of homogeneous PEs provides some spare rows or columns to be used to replace faulty ones.[13] When special diagnostic routines detect damaged elements, the system must be reconfigured according to a specific algorithm.

In PAPIA 2, if a PE is found to be faulty, two adjacent columns are disabled (the one containing the faulty processor and the previous or following column), while two spare columns are activated on the right side of the array. Due to the implementation characteristics of the system, disabling two columns allows array reorganization to take place without modifying the way in which the array works (see Figure 7.6). In order to grant communications between adjacent PEs, indeed, a special short-circuiting hardware is included in each PE and switch element. This is activated by a flag bit.

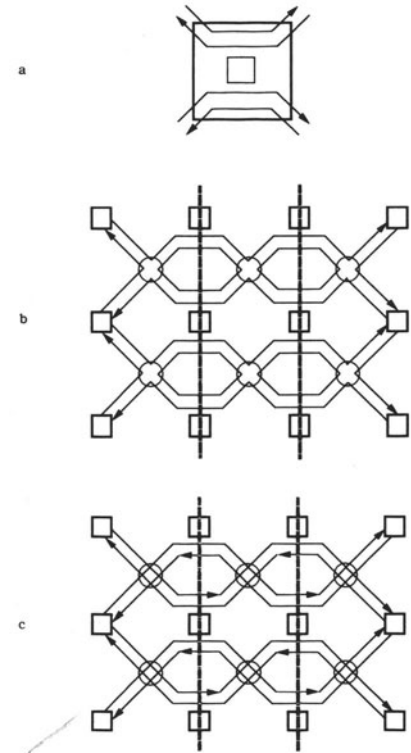


Figure 7.6. Fault management in PAPIA 2. A columns pair substitution for a faulty PE: (a) special local detour required in all PEs of the substituted column pair to reconfigure the array; simplified overall setup after a fault recovery in the horizontal (b) and vertical modalities (c).

When the value of this flag is set, three actions are performed on the pair of columns to be replaced:

1. The PEs are disabled.
2. The PEs and the switch elements are short circuited as shown in Figure 7.6.
3. A special I/O register in each PE (which forms a shift register together with the analogous registers in the PEs of the same row) is short circuited.

The settings of these flags may be provided by a mask which is loaded on the whole array (spare columns included).

d. Simulation Performance Parameters. The usual quantitative analysis is carried out for 8- and 10-level virtual quad pyramids embedded on a flat array of 128×128 PEs. For simulation of the larger pyramid the truncated and complete cases are considered. Regarding the mismatch between the PE array and the pyramid base sizes, the solution adopted is described in Section 7.2.3.1a as follows.

- *Processor load* is equal to 2, $2+2^{k-n+1}$, and 2^{k-n+1} for the 8-level pyramid, 10-level complete pyramid, and 10-level truncated pyramid, respectively, with values of 18 and 16 for the latter two.
- *Expansion,* as for the previous simulation strategy, is $3N^2/(4K^2-1)$ and $3N^4/(4K^2N^2-4K^2)$ for the complete and truncated cases, respectively. These correspond in the case studies to approximately 0.75 and 0.0468 (and the inverse ratio is respectively 1.33, 21.33).
- *Link load* in the diagonal link in the bottom-right quadrant has a value of n, $n2^{k-n+1}+k-n+1$, and $n2^{k-n+1}$, respectively. For the case studies the values are, respectively, 7, 116, and 112.
- *Dilation* in vertical communications is equal to $N/4$, $N/2$, and $N/2$, respectively.

7.2.3.2. Emulating Pyramids in the Polymorphic Torus

The polymorphic torus architecture[8, 9] is a reconfigurable system based on an array of PEs much like the basic mesh. The structure of the PEs is basically standard, as far as processing capabilities are concerned. The polymorphic torus is designed as a massively parallel system and adopts a bit-serial architecture at PE level. Reconfigurability is obtained through a dual interconnection scheme: a physical network (PNET) and an internal network (INET).

The PNET is actually a 4-connected, N×N mesh, augmented with toruslike and/or spiral-like connections at the borders of the array. A PE is located at each junction of the physical mesh, which is interconnected to the PNET through the INET. At each PE, the INET is an internal set of connections among the four ports of the corresponding PNET node. In its most general formulation, the INET node is a complete graph of the four connections (east, south, west, north) of the PNET node.

The novel feature of the system rests with the level of autonomy available at each PE. Within the classes of processor autonomy defined in Section 5.2.2.4 for SIMD systems, the polymorphic torus supports "connection autonomy" as well as the usual "operation autonomy." Indeed, while the physical network is fixed, the internal one is programmable by loading each INET node with proper configuration settings. The specification of these settings can be described through the interconnection function SHORTPORT $\{p_1\},\{p_2\}, \ldots,$ where $\{p_1\}, \{p_2\}, \ldots,$ are the set of connections in the INET that are tied together. For example, the effect of SHORTPORT {N,E} is to directly connect the north and east ports, whereas SHORTPORT {N,S}, {W,E} creates a crossover function that bypasses the PE. Since the settings of the INET nodes can be established locally, this architecture allows us to embed both regular and irregular interconnection patterns among the PEs.

The embedding of various topologies onto the polymorphic torus eventually results in programming the INET at each PE. The approach can be dynamic because each PE performs the functions of possibly more nodes of the target architecture—that is, not only on the basis of its PNET address but also on the basis of the time. The embedding is said to be *processor dependent* if the interconnection function of the INET uses the PE address and time only as arguments. The embedding is *data dependent* if the function takes any data available in the local memory of the PE as its argument, possibly due to the outcome of some previous operation.

A prototype realization of the polymorphic torus concept is YUPPIE (Yorktown ultra parallel polymorphic image engine), described in detail in Ref. 9. In fact, only a restricted subset of the shortporting capabilities of the INET is implemented in YUPPIE. This does not in any way influence the embedding considered here.

With regard to pyramid simulation, the interconnection graph embedded through the INET is regular and preconfigurable. No dependence on local data is necessary except for the address of the PE. The mapping proposed in Ref. 8 for the quad pyramid produces the PN distribution shown in Figure 7.2. In this embedding an important role is played by the shortporting capability, which

substantially reduces dilation. Horizontal communications have dilation 1, hierarchical communications, due to the city-block transmission modality, have dilation 2. The other simulation's performance parameters remain unchanged.

7.3. PYRAMIDS AND HYPERCUBES

A second class of architecture which is becoming more and more popular is that of hypercubes. Since the first proposal by Pease,[14] the family of hypercubes from a topological viewpoint has been investigated in depth in the last two decades. The Boolean n-cube or hypercube[15] can be defined as follows: for n an integer, $n \geq 0$, the n-dimensional hypercube H_n consists of 2^n nodes each labeled with an n-bit number. Two nodes are connected if and only if their binary labels differ in only one bit.

In the following, the embedding of pyramids onto this topology is analyzed on a theoretical basis and discussed in some detail for a family of real system (the Connection Machine models CM1 and CM2).[16]

A family of embeddings of pyramids onto hypercubes tackles this problem by considering the pyramid as a stack of meshes using the known results for *mesh embedding* into the hypercube. A central role in all such solutions is played by the reflexive Gray codes. Following the notation of Stout,[17] let G_d be a Gray code for integers 0–2^d-1: the recursive definition of G_{d+1} from G_d applies the reflexive property, so $G_{d+1}(n) = 0G_d(n)$ if $0 \leq n < 2^d$ and $G_{d+1}(n) = 1G_d(2^{d+1}-1-n)$ if $2^d \leq n < 2^{d+1}$.

Let us now consider the restricted problem of mapping a bidimensional, square mesh of $M \times M$ nodes into an hypercube which has the same number of PEs, PE (i, j) being the coordinates of a generic node for $0 \leq i, j \leq 2^m - 1$, $m = \log_2 M$. The target hypercube will be H_{2m}, which thus has 2^{2m} PEs. The label of the PE which maps node (i, j) can be obtained either by collating $G_m(i)$ with $G_m(j)$, in a *collated gray code,* denoted by $CG_m(i, j)$, or by shuffling the bits of $G_m(i)$ with those of $G_m(j)$ in the *shuffled Gray code,* denoted by $SG_m(i, j)$:

$$
\begin{aligned}
G_m(i) &= i_{m-1} i_{m-2} \cdots i_0 \\
G_m(j) &= j_{m-1} j_{m-2} \cdots j_0 \\
CG_m(i,j) &= i_{m-1} i_{m-2} \cdots i_0 j_{m-1} j_{m-2} \cdots j_0 \quad \text{collated Gray code} \\
SG_m(i,j) &= i_{m-1} j_{m-1} i_{m-2} j_{m-2} \cdots i_0 j_0 \quad \text{shuffled Gray code}
\end{aligned}
\tag{7.4}
$$

Table 7.1. Two mappings derived from Eqs. (7.4).

$i \backslash j$	00	01	11	10
00	0000	0001	0011	0010
01	0100	0101	0111	0110
11	1100	1101	1111	1110
10	1000	1001	1011	1010

Collated Gray Code

$i \backslash j$	00	01	11	10
00	0000	0001	0101	0100
01	0010	0011	0111	0110
11	1010	1011	1111	1110
10	1000	1001	1101	1100

Shuffled Gray Code

The two mappings derived from Eqs. (7.4) are shown in Table 7.1 for $m=2$. As expected, owing to the properties of the chosen code, in both cases, neighboring nodes in the mesh are mapped to PEs whose labels differ by 1. Extending the mapping to pyramids basically involves the use of either code to map a single level of the pyramid in a one-to-one fashion with the targeted hypercube and then including levels below and above.

Alternative solutions, which do not rely on the mesh embedding, are also possible. These are analyzed in Section 7.3.2. Independent of the chosen approach, the central issue in the embedding of pyramids is that of devising a scheme where the neighborhood of a pyramid node is mapped to PEs that are as close as possible, even though not all adjacent. Indeed, it is well known that the basic pyramid cycle, which consists of a node and two children, can never be mapped with dilation 1 onto any cycle in a hypercube which always has an even number of PEs.

7.3.1. Mesh-Based Embeddings

The first embedding of pyramids into hypercubes was proposed by Stout[17] according to the use of the collated Gray code. The first proposed mapping is defined as follows.

Let $N_i = 2^d$ be the number of PEs in hypercube H_d and $K \times K$ the dimension of the base of a pyramid of $k = 1 + \log_2 K$ levels (levels in the pyramid have increasing labels, starting from 0 at the base and reaching $k-1$ at the vertex). The first case considered is when the number of PEs available is smaller than the number of nodes in the base of the pyramid, that is, $N_i < 2^{2(k-1)}$. The mapping associates on a one-to-one basis each pyramid node of level $k-1-d/2$, which consists of an array of $2^{d/2} \times 2^{d/2}$ elements, with each PE in the hypercube. Moreover, the crinkled format is used to store bigger levels of the pyramid, so a PE maps the complete subpyramid of base $2^{k-1-d/2} \times 2^{k-1-d/2}$, consisting therefore of $k-d/2$ levels. By denoting a PN of level h, located at position i, j in the corresponding mesh, with a triplet (h, i, j), the following mapping of the pyramid coordinates to the hypercube address results:

$$CG_d\left(\lfloor i\, 2^{-(k-1-d/2-h)} \rfloor, \lfloor j\, 2^{-(k-1-d/2-h)} \rfloor\right), \qquad 0 \le i, j < 2^{k-1-h}, \quad 0 \le h \le k-1-d/2 \tag{7.5}$$

The mapping of nodes of higher levels is defined by observing that only one quarter of the PEs is used for each successive level. This quarter can be selected on the basis of a fixed value of the last bit of the indexes i and j in Gray code (e.g., the last digit of the associated Gray code 0). The PEs where both digits take on that value are chosen as the parents in the next level for each 2×2 subarray. This selection process is carried out recursively on successive levels, with larger blocks and longer sequences of least significant coincident digits. Denoting by 0^t a string of t occurrences of zeros, the mapping of PN (h, i, j) is

$$G_{k-1-h}(i)0^{h-(k-1-d/2)}G_{k-1-h}(j)0^{h-(k-1-d/2)}, \qquad 0 \le i, j < 2^{k-1-h}, \quad k-1-d/2 < h \le k-1 \tag{7.6}$$

Table 7.2 shows the resulting mapping for the case $k = n+1 = 4$. The properties of this embedding in terms of data movement are quite evident.

Table 7.2. The resulting mapping for the case $k = n + 1 = 4$.

$i \backslash j$	000	001	011	010	110	111	101	100
000	000000	000001	000011	000010	000110	000111	000101	000100
001	001000	001001	001011	001010	001110	001111	001101	001100
011	011000	011001	011011	011010	011110	011111	011101	011100
010	010000	010001	010011	010010	010110	010111	010101	010100
110	110000	110001	110011	110010	110110	110111	110101	110100
111	111000	111001	111011	111010	111110	111111	111101	111100
101	101000	101001	101011	101010	101110	101111	101101	101100
100	100000	100001	100011	100010	100110	100111	100101	100100

a

	000	001	011	010	110	111	101	100
000	3	0	0	1	1	0	0	2
001	0	0	0	0	0	0	0	0
011	0	0	0	0	0	0	0	0
010	1	0	0	1	1	0	0	1
110	1	0	0	1	1	0	0	1
111	0	0	0	0	0	0	0	0
101	0	0	0	0	0	0	0	0
100	2	0	0	1	1	0	0	2

b

Pyramid levels stored in the crinkled format resolve all vertical near-neighbor exchanges and most of the horizontal ones locally in the memory of the PE to which they are mapped. As noted, no possible efficient propagation modality is available with the hypercube, so the emphasis is on obtaining a reduced dilation. Indeed, dilation is very low, since the mapping of pyramid levels above level $k-1-d/2$ keeps neighboring nodes on PEs that are as close as possible. In particular, the 4-connectivity within each level is preserved, and hierarchical connections have a maximum dilation of two links.

The second case to be considered is when the number of PEs of the hypercube is larger than the number of PNs in the base of the pyramid. Since the PNs of levels above the base are one third of those in the base, the hypercube H_d has at least twice as many PEs as PNs in the base of the pyramid; that is, $N_i = 2^d \geq 2^{2k-1}$ [i.e., the expansion parameter is $3 \cdot 2^d/(2^{2k+1}-1)$].

Let us consider the case $d = 2k-1$. The suggested embedding treats the base of the pyramid as a mesh, perfectly embedded in either half of the hypercube. The remaining levels are mapped in the other half by mapping the parents "along the dimensions of the hypercube"[17] (see Table 7.3). Nodes belonging to the first level above the base are mapped in PEs of the second half of the hypercube that are connected to PEs in the first half acting as parents according to the embedding represented in Table 7.2b. That is, candidate parents have been relocated by using one dimension of the hypercube. The process could be iterated for the remaining levels if the hypercube had any other unused dimensions.

Alternatively, note that all PEs working as parents have a least significant digit equal to 0 in their address. The least significant digit is treated as the next dimension and is used accordingly. With reference to Table 7.3, the parent of the four top-rightmost nodes in level 1, labeled 2 and mapped to PE 000100 in H_6 (see Table 7.2), is now mapped to PE 1000101. This mapping gives the least possible processor load; in fact each PE simulates at most one PN. Given that now three hops are necessary for at least one of the children to have its value transmitted to the parent, dilation increases to 3. The load of the links, therefore, drops considerably and attains its minimum of 3, as can easily be seen.

7.3.1.1. Shuffled Gray Code Embeddings

The mappings so far described are based on the collated Gray code. For a pyramid, the alternative use of a shuffled Gray code leads to the following arrangements of pyramid nodes to PEs:

Table 7.3.

	000	001	011	010	110	111	101	100
000	0	0	0	0	0	0	0	
001	0	0	0	0	0	0	0	0
011	0	0	0	0	0	0	0	0
010	0	0	0	0	0	0	0	0
110	0	0	0	0	0	0	0	0
111	0	0	0	0	0	0	0	0
101	0	0	0	0	0	0	0	0
100	0	0	0	0	0	0	0	0

Lower half of the hypercube H_7

	000	001	011	010	110	111	101	100
000	1	2		1	1		2	1
001		3						
011								
010	1			1	1			1
110	1			1	1			1
111								
101								
100	1	2		1	1		2	1

Upper half of the hypercube H_7

$$\begin{aligned} &SG(i, j)4^{h-(k-1-d/2)}, \quad 0 \le i, j < 2^h, \quad k-d/2-1 \le h \le k-1 \\ &SG(\lfloor i2^{h+d/2-k+1} \rfloor \lfloor j\, 2^{h+d/2-k+1} \rfloor)' \quad 0 \le i, j < 2^h, \quad 0 \le h \\ &\qquad < k-k-d/2-1 \end{aligned} \tag{7.7}$$

This mapping is the correspondent to Eqs. (7.5) for the collated Gray code. Between the two mappings there is a graph isomorphism. The parameters that measure the efficiency of the simulation are therefore the same as in the collated Gray code, and nothing special can be added.

With either mapping (shuffled or collated Gray code), the simulation parameters introduced in Section 7.1 can be evaluated as follows. In the two cases considered, the numerical example is given on a H_{10} hypercube simulating the previously considered 10-level pyramid:

- *Processor load* is, as above, just 1 when the number of PEs is greater than the number of PNs; otherwise, at PE_0, which acts for a whole subpyramid and as a single node in each of the remaining $d/2$ levels, it is $(2^{2k-d}-1)/3+d/2$. The least loaded PE only works for the subpyramid, and its load compares fairly well with that of the maximum loaded one. In the case study of a 10-level pyramid simulated by an H_{10} hypercube, *processor load* attains 346, while the smallest value is 341, which shows an extremely well balanced simulation.
- *Expansion* is $3 \cdot 2^d/(2^{2k}-1)$ and becomes 0.003 in the case study. Its inverse 341.33 is almost exactly the smallest value of the processor load, a result to be expected due to the good balancing of the processor load.
- *Dilation* has been shown previously to be 3 and 2, respectively, when the number of PEs is greater or less than the number of PNs, an exceptionally good result if compared with simulations on meshes.
- *Link load* is 3 when the number of PEs is greater than the number of PNs; otherwise it is $2^{k-1-d/2}+d/2-3$ between a subset of PEs which map the 4×4 level in the pyramid. This subset consists of the eight couples of PE mapping nodes which are adjacent but not siblings. Indeed, as a result of the mapping, the link between each pair of these PEs sustains all lateral connections between the subpyramid of base $2^{k-1-d/2}\times2^{k-1-d/2}$ and the lateral connections of the nodes in the higher levels they host, which are $d/2-2$. The resulting link load value for the case study is 18.

7.3.2. Other Embeddings

When the hypercube simulating the pyramid is large enough to give rise to a processor load equal to 1 (the second case just analyzed), the embedding

of the pyramid into the two halves of the hypercube (one for the base, the other for the remaining levels) is not the only possible solution. Two classes of new mappings have been produced;[18] one attains the same values for dilation and link load as the second embedding above (namely, dilation 3 and link load 2), and the other optimizes dilation (which is lowered to 2) at the expense of link load (which increases to 3). A short description of such proposals follows; since the details are rather complex, the interested reader is referred to the original paper. The first case is presented, even if it is intrinsically equivalent to the analogous result by Stout, because the same performance is obtained without an explicit mapping of any submesh in the pyramid. It also serves as an introduction to the second one, which is peculiar by itself because the decrease in dilation is obtained at the expense of the communication among siblings in the pyramid. Indeed, while in all the other solutions the longest sequence of hops (dilation is 3) in the hypercube necessary to simulate a single connection in the pyramid involves a parent node and one of its children, in this case the longest one (dilation is 2) is attained also among nodes adjacent within a level of the pyramid.

7.3.2.1. Optimum Link Load Embedding

The algorithm defines the mapping g_k: $P_k \rightarrow H_{2k+1}$ of a pyramid of $k+1$ levels onto the minimum-sized hypercube, yielding a processor load of 1, thus having twice as many PEs as there are nodes in the base of the pyramid. The mapping g_k is illustrated in Figure 7.7 for $k=1$ (a two-level pyramid). The nodes of the pyramid are depicted as black circles. Their label (h, i, j) is associated with the corresponding hypercube address. Dotted circles denote those PEs in H_3 which do not map any PN, but are used to connect the parent PN to its children. Note that this mapping is one of 24 equivalent allocations.

The mapping clearly shows that the connection between pyramid nodes **000** and **111** (parent and southeastern child) has dilation 3 because it involves two intervening PEs (100 and 101). Furthermore, the hypercube link between PE 100 and PE 000 is the maximum loaded one and produces a link load 2 (drawn twice in the figure). For the recursive construction of larger pyramids, it is essential that the apex of the pyramid be mapped to a PE (in this case 000) adjacent to another PE (in this case 001) which does not map any other PN. The link load between these two PEs must be a minimum. In the following discussion, these two PEs will be referred to as α and β, respectively.

Figure 7.7 shows three more embeddings of P_1 into H_3. They can be derived from g_1 by symmetrical reflections: $g_{1,v}$ is the vertical reflection with respect to a plane passing through the apex of the pyramid and splitting the

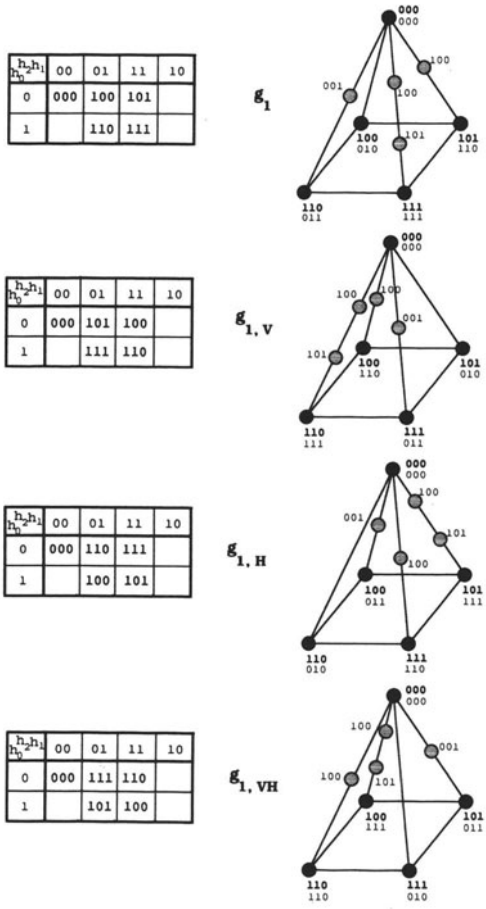


Figure 7.7. One of the 24 equivalent embeddings $g_1 : P_1 \rightarrow H_3$, together with the three associated rotations $g_{1,V}$; $g_{1,H}$; and $g_{1,VH}$. In the drawings, pyramid nodes are black circles and the label (h, i, j) is bold; PEs in H_3 have labels $h2$ $h1$ $h0$ in plain text; those used only for message routing are indicated by dotted circles.

base into western and eastern children. $g_{1,H}$ is the horizontal reflection with respect to the plane through the apex splitting the base into southern and northern children. $g_{1,VH}$ is the composition of $g_{1,V}$ and $g_{1,H}$. The role of these alternative embeddings is crucial in the recursive construction of g_k: $P_k \rightarrow H_{2k+1}$ from g_{k-1}: $P_{k-1} \rightarrow H_{2k-1}$. This will be explained in the case of the mapping of the next larger pyramid P_2 into H_5.

Basically, P_2 can be seen as the assembly of four instances of the smaller pyramid P_1, each associated with one of the nodes of level 1 in P_2, plus a

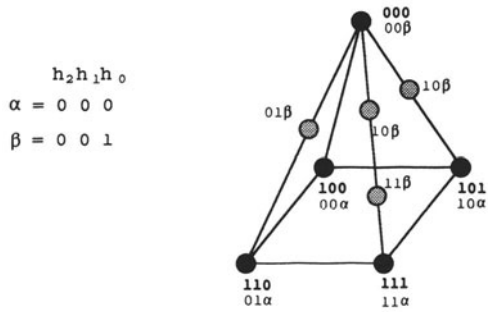


Figure 7.8. Recursive construction of a pyramid of $k+1$ levels P_k onto a hypercube H_{2k+1} with dilation 3 and link load 2. The top subpyramid of P_2 is mapped in a three-dimensional subcube of H_5 using g_1.

further level consisting of the apex alone. In the same way, H_5 is made up of four smaller hypercubes H_3, namely $00H_3$, $01H_3$, $10H_3$, $11H_3$. The desired mapping g_2: $P_2 \rightarrow H_5$ is arrived at as follows: the subpyramid P_1 emanating from node **100** of P_2 is mapped to $00H_3$ using g_1, the subpyramid P_1 emanating from node **110** of P_2 is mapped to $01H_3$ using $g_{1,H}$, the subpyramid P_1 emanating from node **101** of P_2 is mapped to $10H_3$ using $g_{1,V}$, and finally the subpyramid P_1 emanating from node **111** of P_2 is mapped to $11H_3$ using $g_{1,HV}$. The use of reflections guarantees the required adjacency between nodes in the base of the target pyramid, that is, level 2 of P_2.

The mapping of the topmost subpyramid in P_2 is obtained in the three-dimensional subcube of H_5 generated by $\{00\alpha, 01\alpha, 10\alpha, 11\alpha, 00\beta, 01\beta, 10\beta, 11\beta\}$ using the property of the basic embedding g_1 that α is adjacent to β. Note that this solution is not unique. With reference to Figure 7.8, the topmost subpyramid of P_2 is mapped using g_1 in such a way that the resulting final embedding g_2 preserves the desired properties of a dilation of 3, link load of 2, and the required condition for the recursiveness of the algorithm (the apex of P_2 is mapped to the PE labeled 00β adjacent with no congestion to PE 01β). Figure 7.9 shows the embedding g_2 of the nodes of P_2 laid out onto a planar representation of H_5 along with the complete three-dimensional representation.

7.3.2.2. Optimum Dilation Embedding

A minimization of the dilation is obtained by using a different type of embedding of P_1, depicted in Figure 7.10, denoted by f_1. Its main characteristics are that any two nodes in P_1 are connected by at most one intervening PE (only three couples are directly linked with dilation 1) and that two links in H_3

h_4h_3 \ h_2h_1	00	01	11	10
00	100	200	201	
01	110	230	231	
11	111	233	232	
10	101	203	202	

$h_0=0$

00	01	11	10
000	210	211	
	220	221	
	223	222	
	213	212	

$h_0=1$

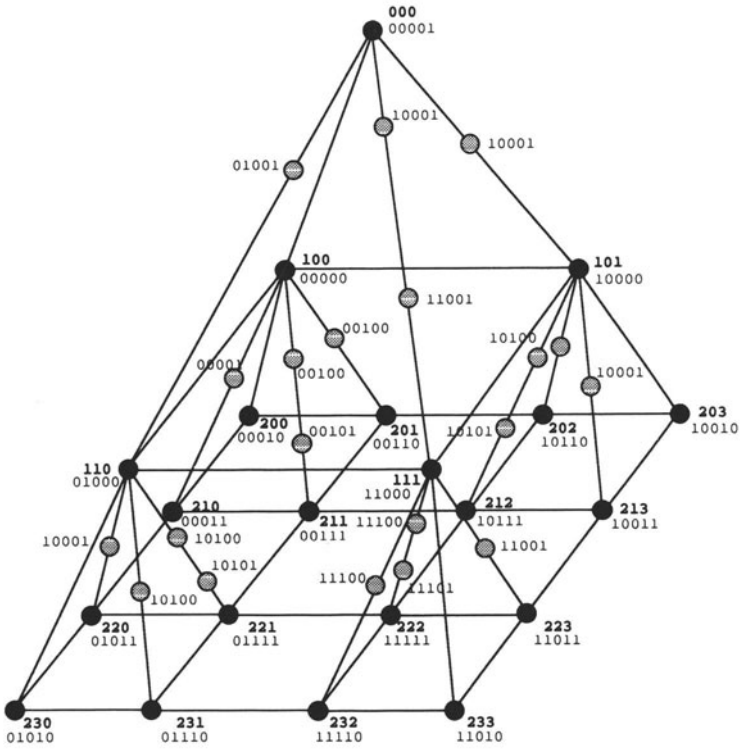
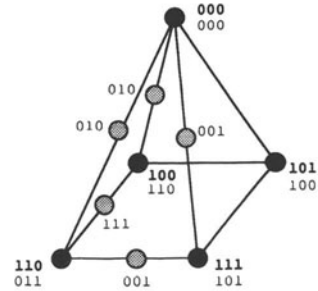


Figure 7.9. The complete embedding of P_2 into H_5 is obtained using the four versions of P_1 of Figure 7.7 to extend the topmost subpyramid shown in Figure 7.8 to the third level. By construction, the resulting embedding maintains dilation 3 and link load 2. The mapping of the 21 pyramid nodes of P_2 onto the 32 PEs of H_5 is shown both in a table and in a drawing.

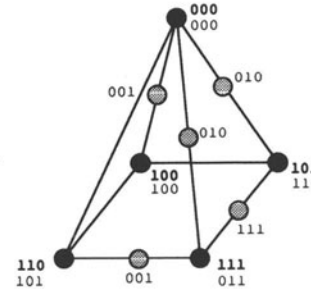
h_2h_1 / h_0	00	01	11	10
0	**000**		**100**	**101**
1		**110**		**111**

f_1



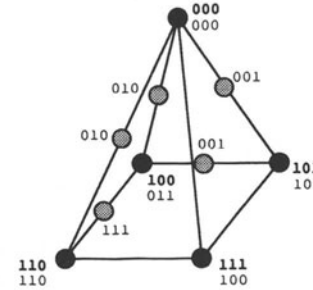
h_2h_1 / h_0	00	01	11	10
0	**000**		**101**	**100**
1		**111**		**110**

$f_{1,V}$



h_2h_1 / h_0	00	01	11	10
0	**000**		**110**	**111**
1		**100**		**101**

$f_{1,H}$



h_2h_1 / h_0	00	01	11	10
0	**000**		**111**	**110**
1		**101**		**100**

$f_{1,VH}$

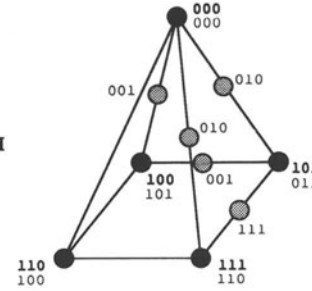


Figure 7.10. One of the 24 equivalent embeddings $f_1 : P_1 \rightarrow H_3$, together with the three associated rotations $f_{1,V}$; $f_{1,H}$; and $f_{1,VH}$. In the drawings, pyramid nodes are black circles and the label (h, i, j) is bold; PEs in H_3 have labels $h_2\ h_1\ h_0$ in plain text; those used only for message routing are indicated by dotted circles.

have congestion 2. Thus, this elementary embedding attains the desired reduction of dilation to 2.

It is then used, along with its reflections $f_{1,V}$, $f_{1,H}$, and $f_{1,VH}$ built in accordance to the reflections of g_1 (also shown in Figure 7.10), in a recursive algorithm. This produces in a top-down fashion the mapping f_k of P_k into H_{2k+1} by first using f_1 for the topmost subpyramid of P_k. The recursion applies f_1 for each new subpyramid whose apex (h, i, j) has $i_{\text{mod } 2} = j_{\text{mod } 2} = 0$; $f_{1,V}$ for $i_{\text{mod}2} = 0$, $j_{\text{mod } 2} = 1$; $f_{1,H}$ for $i_{\text{mod } 2} = 1$, $j_{\text{mod } 2} = 0$; and $f_{1,VH}$ for $i_{\text{mod } 2} = j_{\text{mod } 2} = 1$. The mapping f_2 of the three-level pyramid P_2 is shown in Figure 7.11 both in a table representation of H_5 and in the three-dimensional sketch.

The recursive use of f_1 and of its reflections guarantees that dilation is kept at 2. As for the link load, the algorithm gives rise to a few instances of links among PEs which receive congestion 3, even if no such link exists within the basic embedding f_1. By inspection, one can verify that f_2 contains two such links, namely (00001–00101) and (10001–10101).

Details of the embeddings (both g_k and f_k), along with analytic derivations of the average link load and dilation, can be found in Ref. 18. While the quantitative assessment clearly shows where the optimum is in terms of the defined parameters, another type of assessment can be made. This regards the homogeneity of the communication pattern. Mesh-based embeddings naturally map communications primitives for intralevel data exchanges, and do so without introducing any unbalances. Vertical data exchanges are slower, because of the larger dilation, and more irregular, because only some of the pyramid edges have the dilation value (maximum). The embeddings introduced in this section not only do not use the mesh, but also introduce irregular communication patterns between nodes of the pyramid belonging to the same logical level. This is the cost to be paid in reducing the dilation.

7.3.3. Embeddings on Real Systems

The description of the mappings of pyramids on the hypercube architecture has not only a theoretical interest. The hypercube structure has been adopted in quite a few commercial multiprocessor and multicomputer systems. NCUBE, Intel, Floating Point Systems, and Thinking Machines have built and distributed hypercube machines of different sizes and conceptions. Other companies, such as MEIKO, offer multiprocessor platforms, mainly based on the transputer, that can be configured as hypercubes.

The discussion of the mapping made so far, although potentially useful

h_4h_3 \ h_2h_1	00	01	11	10
00	000		100	101
01	213	210		
11			200	203
10	212	211	201	202

$h_0=0$

00	01	11	10
	110		111
223		220	
	230		233
222	231	221	232

$h_0=1$

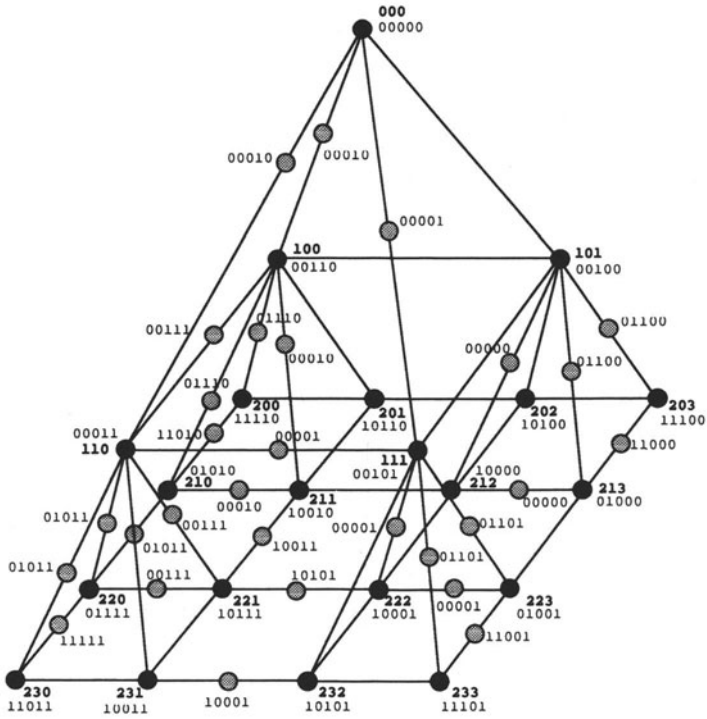


Figure 7.11. A complete embedding of P_2 into H_5 obtained using the four versions of P_1 of Figure 7.10. The recursive construction applies the four reflections of f_1 according to the values of the least significant bit of i and j in nodes $(1, i, j)$. By construction, the resulting embedding has dilation 2 and link load 3.

for all hypercube machines, is particularly relevant for fine-grained ones, since they comply best with the compact pyramid structure. The family of systems by Thinking Machines, including the models CM-1 and CM-2, is the closest to the massively parallel fine-grained model. It has been extensively studied and a few proposals of practical embeddings have produced actual running environments to develop pyramid algorithms. After a description of the features of the CM architecture and software environment pertinent to this discussion, three embeddings will be described for the quad pyramid mapping.

Furthermore, the bin-pyramid architecture adopted by the SPHINX system (described in Chapter 5) has been fully simulated on the CM-2 as well. This effort is the only attempt to study the special embedding of a bin pyramid on the hypercube, and it is reported here to complete the subject.

7.3.3.1. The Connection Machine Environment

The Connection Machine (models CM-1 and CM-2) is a SIMD multiprocessor system consisting of up to 64k-bit serial PEs. The PEs are integrated into chips, each containing 16 PEs, usually arranged as a 4×4 small mesh. The chips are interconnected to form a hypercube of 12 dimensions, H_{12}, by the *router* subsystem; each PE in the machine is therefore identified by a single 16-bit address, and the whole set of physical processors results in a 16-dimension hypercube H_{16}. The model CM-1 had a NEWS communication network that created a single two-dimensional mesh by interconnecting the 4×4 small grids embedded in each chip. Such a network is not available in the CM-2 model, which resolves this type of communication with an optimized use of the router.

The software environment in the CM supports the notion of "virtual processor." An application can be coded as if the number of processors were larger than what is really available. Each physical processor takes care of a number of such virtual processors by time-slicing through its allocated subset. The ratio of virtual processors to physical processors is called the *vp ratio* (VPR). The mapping of virtual processors to physical PEs is somewhat constrained, given that both must be a power of 2. Moreover, the data-parallel programming style is supported in the CM by the notion of *vp set*. A vp set is a group of virtual processors arranged logically in a specified *geometry*, which is an n-dimensional grid. The cardinality of the vp set must be an integer multiple of the physical dimension of the machine. Furthermore, the geometry must have a power-of-2 dimension on each axis in the n-dimensional grid.

Each virtual processor is assigned a "send address," which is unique

within the vp set. It is an unsigned integer extending from 0 to the cardinality of the vp set minus 1. The send address can be broken down into two main fields; the physical address (most significant bits) and the virtual part (least significant bits). The first specifies the physical PE that maps the virtual processor, and the virtual part identifies the virtual processor within the memory of the physical one. The physical part can be further divided into ''on-chip bits'' (four) and ''off-chip bits'' (the dimension of the hypercube of chips). As an example, a 512×512 mesh mapped onto the largest CM-2 has a VPR of 4. The ''send address'' consists of 18 bits, of which the least significant two are the virtual component. The next 4 identify each PE within the chip, and the remaining 12 are the address of each chip in H_{12}.

The mapping of a geometry onto the physical machine is only partially under program control. If nothing is specified, the system allocates the n-dimensional mesh in the machine. Therefore any two neighbors in the mesh are either in the memory of the same physical processor (coincident physical address) or on physical processors on the same chip (coincident hypercube address) or on chips directly connected through the router by a single link (hypercube addresses with a Hamming distance of 1). The Gray code is used in the same way when handling the geometry. The physical component of the send address is split into as many parts as are the axis of the geometry. Each part codes the corresponding axis index with a reflexive Gray code. The resulting coding is therefore a collated Gray code. A more detailed control of the virtual-to-physical mapping allows us to specify the relative weights of the axis of the geometry. Mesh locations belonging to the axis which sustains the highest communication load are mapped in physical contiguity on the PEs.

a. CM-1 Embedding. The first pyramidal programming environment was designed to run on the CM-1 at Columbia University.[19] It did not rely on the virtual environment of the later models of the Connection Machine. Yet, it managed to simulate pyramids with a base of up to 256×256 nodes, explicitly controlling the allocations of nodes in higher levels so that a PE acts at most for one extra node above the base (processor load 2). The mapping favors vertical communications against horizontal ones: vertical dilation is 3; the horizontal one scales with the dimensions of the hypercube (16 in the largest CM-1 model).

The mapping of nodes in the base of the simulated pyramid relies on a shuffled row-major indexing scheme, denoted by SRM(i, j) in the following. It uses the standard bidimensional mesh supported in hardware by the CM-1 to perform near-neighbor data exchanges in unit time.

Nodes in higher levels are mapped to PEs whose send address is given by the expression

$$[1+\mathrm{SRM}_{d-2h}(i, j)]4^h - 2^{h-1}, \quad 0 \le i, j < 2^{k-1-h}, \quad 1 \le h \le k-1 \tag{7.8}$$

Table 7.4 shows the mapping of P_3 onto H_6. Clearly, each PE works for a maximum of two nodes in the pyramid. On each level, the positions of nodes within the corresponding mesh comply with the shuffled row-major sequence. Despite this, the embedding of such meshes on the hypercubes does not guarantee adjacency among neighbors. Therefore, logical neighbors are not located on adjacent PEs on the hypercube.

As to horizontal communications, a single shift of data along any of the four cardinal directions of the mesh requires a minimum of $d/2-1$ transmission steps over hypercube links. This is easily shown if we consider the movement of data in the first level above the base ($h=1$) in an eastward direction. All couples of nodes $(1, i, j)$ which are located on the two central columns $j_1 = 2^{k-3}-1$ and $j_2 = j_1+1$ have shuffled row-major indexes $\mathrm{SRM}(i, j_1)$ and $\mathrm{SRM}(i, j_2)$ which differ in all even positions. The "send addresses" resulting from the mapping of Eq. (7.8) differ accordingly at the same positions, excluding bit position 0 (it is assumed that the least significant bit is labeled 0). With reference to Table 7.4, sending data from nodes $(1, i, 1)$ to nodes $(1, i, 2)$ requires first using hypercube dimension 4 (the left half of the mesh is transmitted to the right half), then dimension 2 (within the right half of the mesh the right half is sent to the left one). It can be shown that these $d/2-1$ transmission steps on the hypercube cover all the movements, on all levels, one position eastward on the meshes above the base.

Regarding vertical communications, each transition by one level requires three steps over the hypercube links (dilation 3). The upper-left child of a node sends its data to its east neighbor, which in turn routes it southward. Finally the parent node is reached through a third movement (this step is missing only between the base of the pyramid and level 1). The hypercube dimensions involved in the data transfer between levels h and $h+1$ are, respectively, $2h$, $2h+1$, and $h-1$. A SIMD transfer among all couples of levels is possible and takes at most three steps. Data transfer for two children takes two steps (precisely over hypercube dimensions $2h+1$ and $h-1$ for the upper right child and over $2h$ and $h-1$ for the lower left one). The last child is at a unit distance along dimension $h-1$.

b. CM-2 Embeddings. Two more environments for pyramidal processing[20, 21] have been set up on the second major version of the Connection

Table 7.4. Mapping of P_3 onto H_6

$i\backslash j$	000	001	010	011	100	101	110	111
000	000000	000001	000100	000101	010000	010001	010100	010101
001	000010	000011	000110	000111	010010	010011	010110	010111
010	001000	001001	001100	001101	011000	011001	011100	011101
011	001010	001011	001110	001111	011010	011011	011110	011111
100	100000	100001	100100	100101	110000	110001	110100	110101
101	100010	100011	100110	100111	110010	110011	110110	110111
110	101000	101001	101100	101101	111000	111001	111100	111101
111	101010	101011	101110	101111	111010	111011	111110	111111

a

	000	001	010	011	100	101	110	111
000	0	0	0	0	0	0	0	0
001	0	1	0	1	0	1	0	1
010	0	0	0	0	0	0	0	0
011	0	1	2	1	0	1	2	1
100	0	0	0	0	0	0	0	0
101	0	1	0	1	0	1	0	1
110	0	0	0	0	0	0	3	0
111	0	1	2	1	0	1	2	1

b

Machine, the CM-2. Both take advantage of the support of the virtual processor concept but differ in their allocation strategy.

The embedding of Sher and Rosenfeld[20] uses the shuffled Gray code to map virtual processors onto physical PEs. The send address of the PE simulating pyramid node (h, i, j) is given by

$$SG_{2k-2-2h}(i, j)\, 4^h, \qquad 0 \le i.\, j < 2^{k-1-h}, \quad 0 \le h \le k-1 \tag{7.9}$$

The virtual component of this address takes $2k-2-$d bits on a CM-2 configured as an H_d hypercube. For example, a 10-level pyramid ($k=10$) simulated on a maximum configuration CM-2 (which is the hypercube H_{16}) yields a VPR of 4 on the base of the pyramid. Of the 18 bits of the send address, 2 are used to identify each node of the four mapped in the memory of the same PE. The virtual processor scheme produces the crinkled storage format introduced in Section 7.2.1.1.

Equation (7.9) is closely related to Eq. (7.7). Indeed, Eq. (7.7) produce the physical component of the send address of a virtual processor in the Connection Machine environment.

The mapping thus obtained is functionally equivalent to the first mesh-based embedding introduced in Section 7.3.1, described by Eqs. (7.5) and (7.6) and illustrated in Table 2b. Except for the use of the Gray code type, the two solutions are equivalent. Physical processor 0 simulates k nodes, and all other quantitative parameters remain unchanged.

As to data exchanges between neighbors on the pyramid, determining the address of the parent is very easy. For any node (h, i, j) the send address need only be modified by setting bit positions $2h$ and $2h+1$ to zero. Indeed, the parent node of (h, i, j) is labeled $(h+1, \lfloor i/2 \rfloor, \lfloor j/2 \rfloor)$ and has the send address $SG(\lfloor i/2 \rfloor, \lfloor j/2 \rfloor)4^{h+1}$. For an instance, in the mapping of P_9 the send address of a node on the seventh level $(6, i, j)$ is represented by the bit sequence $i_2 j_2$ $i_1 j_1$ $i_0 j_0$ 00 00 00 00 00 00 and the corresponding bit sequence of $(7, \lfloor i/2 \rfloor, \lfloor j/2 \rfloor)$ is i_2 j_2 $i_1 j_1$ 00 00 00 00 00 00 00, where bit positions 12 and 13 have been set to 0. Conversely, the send addresses of the four children are built by setting bit positions $2h-1$ and $2h-2$ to each of the bit couples 00, 01, 10, and 11.

Thanks to the Gray code, only one bit in the send address of a node needs changing for near-neighbor connections within a level. However, the bit to be changed depends on the position of a node within the mesh. With VPR = 1, half the dimensions of the hypercube must be used for a one-step data transfer along one direction in the mesh of the level.

The embedding of Mzaik and Jagadeesh[21] overcomes both the poor near-neighbor communications performance of the CM-1 solution and the uneven usage of the virtual processor scheme of the Sher and Rosenfeld approach.

The mapping handles pyramid levels differently according to the ratio of the corresponding mesh with the CM-2 hypercube. With the usual terminology, a pyramid P_k of $K \times K$ nodes in the base is mapped on the H_d CM hypercube by using a different vp set for each level h with a mesh larger than the hypercube, $0 \leq h < k-1-d/2$. The virtual processor allocation can be set up so that each PE stores in its memory a complete subpyramid (the crinkled storage format). The VPR is uniform throughout the PEs and the vp set of level 0 is the most time consuming that is simulated.

The level of the pyramid containing as many nodes as the PEs of the hypercube ($h = k-1-d/2$), and all other higher levels ($h \leq k-1$) are mapped into a single vp set according to a scheme that is identical to the flat pyramid approach described in Section 7.2.3.1. The first level is mapped using either of the standard mappings of meshes with VPR = 1 (the default is the collated Gray code at chip component level). The meshes *(h, i, j)* of the other levels are relocated by the coordinate transformation given in Eq. (7.2).

The overall VPR in this vp set is 2. All considerations made for the flat pyramid embedding onto a mesh are valid here, as far as higher levels are concerned. Indeed, for the lowest levels the crinkled solution is superior because CM does not support parallel neighbor access.

However, the addressing scheme in the resulting pyramid simulation requires different mechanisms in each of the two sections of the pyramid. Vertical communications below level $k-1-d/2$ are implemented through a CM software primitive named "cross-vp-move." This determines mesh coordinate transformations within different geometries. On the higher levels (that is, within the last vp set), explicit handling of node coordinates is required to build the send address of a parent node. Horizontal communications are similarly managed. The power-of-2 near-neighbor data exchanges required in higher levels are directly supported by the CM hardware and are executed in constant time.

c. The Embedding of a Bin Pyramid. The majority of effort to obtain efficient embeddings onto an hypercube for pyramids has only considered the quad-pyramid topology. Overlapped pyramids have been partially addressed in the work of Sher and Rosenfeld. Other pyramids have not been considered. The exception is the bin pyramid, as defined in Chapter 3. Its motivation is the need to produce a simulation environment for the SPHINX machine (see Chapter 5), which is indeed a bin-pyramid system.

Various possibilities[22] have been analyzed, with the goal of obtaining an architectural embedding of the SPHINX system onto a CM-2 model, usable for effective SPHINX code development down to bit-serial register transfer level. Two approaches are reported here; the first is a graph-theoretical solution usable on any massively parallel hypercube machine, and the second exploits the peculiar software architecture of the CM to arrive at a very efficient embedding.

From a graph-theoretical point of view, a bin pyramid with $K \times K$ nodes in the base has $2K^2 - 1$ nodes and $2k - 1$ levels, with $k = 1 + \log_2 K$. The coordinates of a node (h, i, j) are defined, with the assumption that h is 0 in the apex, so that the following relations hold:

$$0 \le h < 2K - 1, \qquad 0 \le i < 2^{\lfloor (h+1)/2 \rfloor}, \qquad 0 \le j < 2^{\lfloor h/2 \rfloor} \tag{7.10}$$

Any embedding with processor load 1 therefore requires a hypercube with $2k - 1$ dimensions, H_{2k-1}. One half of the hypercube maps the base of the pyramid, the other all the remaining levels. The resulting expansion is roughly 1 (to be precise $(2K^2 - 1)/2K^2$). An embedding is possible that guarantees a link load 2 and a dilation 2, which is the theoretical minimum for any bin pyramid on a hypercube. It is defined by labeling the nodes of the pyramid in top-down fashion according to the following algorithm:

$$\begin{aligned} &\text{label(apex)} = 1 \\ &\quad \text{label}(h+1, i, j) = \text{label}\ (h, \lfloor i/2 \rfloor, \lfloor j/2 \rfloor) + 2^{h+1} \\ &\qquad\qquad \text{or} \\ &\qquad\qquad \text{label}\ (h, \lfloor i/2 \rfloor, \lfloor j/2 \rfloor) + 2^{h+1} - 2^h \end{aligned} \tag{7.11}$$

The choice in the label of the two children of a given node can be used to guarantee adjacency between neighbors on the children levels. Indeed, this labeling scheme maps a node and one of the two children (the one labeled according to the first choice) on adjacent hypercube PEs. At the same time, it maps the two children on adjacent PEs. Therefore the second child is at a Hamming distance of 2 from the parent. Adjacency on the hypercube among nodes that are not siblings depends on the association of the two possible labels between the right and left children. If the association is static—that is, if the right child always takes its label from the former choice and the left one from the second—adjacency within a level is not maintained. Indeed, it can be easily shown that, in such a case, the above algorithm can be replaced by the closed-

form addressing scheme for node (h, i, j) defined as in (7.11), valid in hypercube H_{2k-1}:

$$2^h + \text{RSRM}(i, j) \tag{7.12}$$

In the expression, RSRM(i, j) indicates the reversal, bit by bit, of the shuffled row-major code SRM(i, j).

To preserve adjacency among nonsiblings nodes, it is necessary to resort to a reflexive coding, such as the shuffled Gray code. The addressing scheme becomes

$$2^h + \text{RSG}(i, j) \tag{7.13}$$

where RSG(i, j) indicates the bit by bit reversal of shuffled Gray code SG(i, j).

Figure 7.12 shows the labeling of a bin pyramid of five levels having a 4×4 base according to Eq. (7.13).

The implementation onto the CM of this embedding has two drawbacks. Communications within a level use half of the dimensions of the hypercube, and no correspondence exists among levels, hypercube dimensions, and near-neighbor direction. Furthermore, the mapping is based on the use of a single vp set. Therefore, operations carried out on a single level of the pyramid involve the set of virtual processors of the whole pyramid, with a huge waste of simulation time. This is peculiarly bad if one considers that SPHINX is a multi-SIMD system.

A second embedding which overcomes these problems relies heavily on the storage management of the CM. The approach is to use a standard mesh embedding for each level of the pyramid. This is to preserve near-neighbor adjacency, using a single vp set for each layer (at least for levels larger than a half of the hypercube). This solution is then slightly modified to exploit the virtual-to-physical processors mapping in order to guarantee that a parent and its two children are mapped in the same PE and that variables allocated on each couple of children are mapped on adjacent words in the memory of the physical processor. In such a case, it is possible to use the ''aliasing'' feature of the PARIS language, the assembly-level programming environment of the CM. This allows us to consider two consecutive words in the memory of a PE, each p bits long, as a single $2p$-bit word in a different geometry. So, transmitting two children values to the parent node is simply a different way of addressing the physical memory of the PE. The actual management of the aliases is slightly different if the level of the children is odd or even. The odd case is more efficient, because the even one requires some true movements of data.

h_4h_3 \ h_2h_1	00	01	11	10
00		100	201	200
01	300	301	311	310
11	401	402	412	411
10	400	203	413	410

$h_0=0$

00	01	11	10
000	110	211	210
330	331	321	320
431	432	422	421
430	433	423	420

$h_0=1$

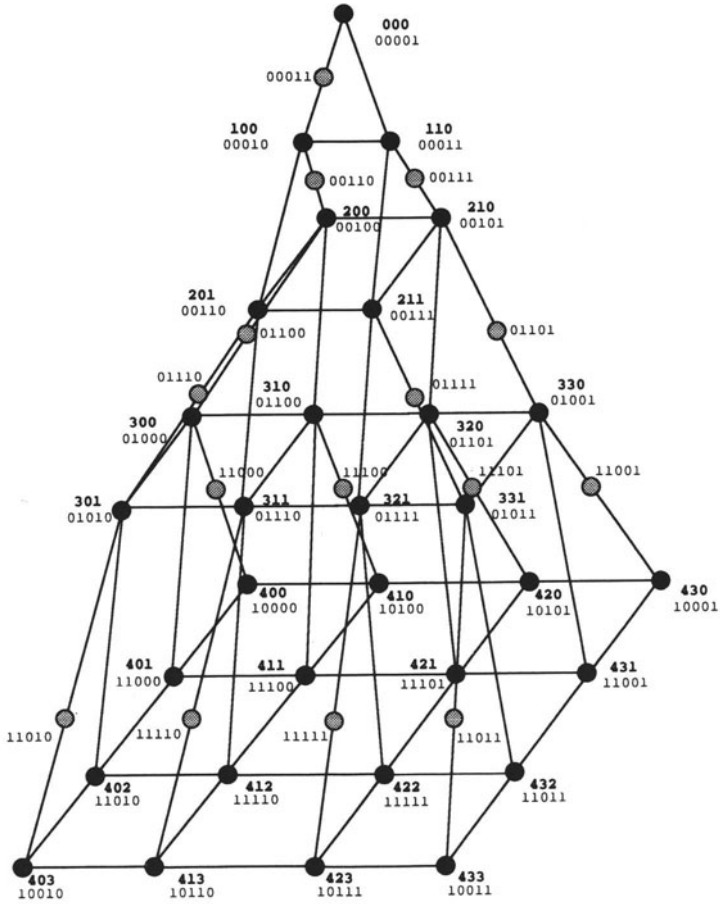


Figure 7.12. Embedding of a five-level bin pyramid into H_5. The two tables indicate the mapping of PNs (in bold) onto the PEs. The drawing highlights the fact that the embedding has dilation and link load 2.

The implementation based on the "aliasing" feature exhibits a considerable speedup (10–20) over the previous solution, but cannot be used on systems that do not support this feature.

7.3.4. The Neighbor Addressing Scheme

The neighbor addressing scheme within a mapping $P \rightarrow H$ consists of the procedure which returns the hypercube address of the PE embedding the chosen neighbor of a pyramid node PN. Although less important than the other quantitative measures of efficiency defined in Section 7.1, it gives some hints to the ease of practically coding the embedding. This can be a relevant problem if the targeted hypercube is a SIMD machine rather than a MIMD one.

Due to the symmetry in the pyramid, only three cases need to be considered, namely, addressing a neighbor within the same pyramid level, addressing the parent, and addressing one of the children:

$$(h, i, j) \rightarrow \begin{matrix} (h, i \pm 1, j \pm 1) \\ (h-1, \lfloor i/2 \rfloor, \lfloor j/2 \rfloor) \\ (h+1, 2i, 2j) \end{matrix} \tag{7.14}$$

Only the first case will be analyzed; analogous considerations are also valid for the others. In the two mesh-based embeddings considered in Section 7.3.1, the addressing procedure requires using $d/2$ hypercube dimensions in H_d on the base of the pyramid on any neighborhood direction. This decreases linearly with the level, since at each new pyramid level two more bits in the addresses are set to a fixed combination of values. Since the embedding of the meshes which correspond to the pyramid levels has dilation 1, the addressing procedure cost is 1 if the PEs have addressing autonomy. Each PN sends this data to its targeted neighbor over the correct hypercube link with no congestion. If, however, the hypercube works under the SIMD model, the cost can rise to $d/2$. Indeed, the changes in the hypercube dimensions can only be carried out sequentially on different sets of PEs.

When the embedding is not explicitly based on the mesh, a more irregular pattern is necessary. For any given neighborhood direction, the required hypercube dimension changes with the level of the pyramid. For example, let us consider a transmission from (h, i, j) to $(h, i+1, j)$ in the embedding of Figure 7.9. On level $h=1$, such a transmission is along h_3, on level $h=2$ it is along h_0 and h_3. The addressing becomes very irregular when it involves nonuniform costs of transmission along the same direction, as in Figure 7.11. Since dilation

raises to 2, even for data exchanges within the same level in the pyramid, at least two hypercube dimensions are always involved. The same case we have just considered now requires three of the five dimensions of H_5.

The implementations on the Connection Machine considered in Section 7.3.3.1 show the relevance of this problem.

7.4. CONCLUSIONS

This chapter has highlighted the capabilities of the two most common massively parallel architectures, the mesh and the hypercube, in emulating compact fine-grained pyramids.

A number of quantitative parameters have been used to measure the efficiency of the embeddings that have been proposed. This type of benchmarking, though rather crude, offers a first-order criterion to select the most convenient solution, according to the peculiarities of the actual system available.

The programming effort implied by the simulation has also been partially analyzed. The near-neighbor access, which is a crucial feature in compact pyramids, has been considered in some detail.

In the case of the mesh topology, there are a few hardware enhancements to the basic structure of the mesh that considerably reduce the cost of the embedding. Notably, the ''flat pyramid'' solution seems quite a good compromise between system complexity and simulation performance.

REFERENCES

1. Q. F. Stout, Mapping vision algorithms to parallel architectures, *Proc. IEEE* **76**(8), 982–995 (1988).
2. H. Li and Q. Stout (eds.), *Reconfigurable Massively Parallel Computers,* Prentice-Hall, Englewood Cliffs, NJ (1991).
3. A. P. Reeves, Pyramidal algorithms on processor arrays, in *Pyramidal System for Computer Vision* (V. Cantoni and S. Levialdi, eds.), pp. 195–214, Springer-Verlag, Berlin (1986).
4. C. Lunghi: Progetto di un chip multiprocessore per macchina piramidale, *thesis,* Pavia University, Italy, 1987.
5. M. J. B. Duff, Pyramids: expected performance, in *Pyramidal System for Computer Vision* (V. Cantoni and S. Levialdi, eds.), pp. 59–74, Springer-Verlag, Berlin (1986).
6. V. Cantoni and S. Levialdi, Multiprocessor computing for images, *IEEE Proc.* **76**(8) (1988).
7. M. G. Albanesi, V. Cantoni, U. Cei, M. Ferretti, and M. Mosconi, Embedding pyramids into mesh arrays, in *Reconfigurable Massively Parallel Computers* (H. Li and Q. Stout, eds.), pp. 123–140, Prentice-Hall, Englewood Cliffs, NJ (1991).

8. H. Li and M. Maresca, Polymorphic-torus architecture for computer vision, *IEEE Trans. Pattern Anal. Machine Intell.* **11**(3), 233–242 (1989).
9. M. Maresca and H. Li, Polymorphic VLSI arrays with distributed control, in *Reconfigurable Massively Parallel Computers* (H. Li and Q. Stout, eds.), pp. 33–63, Prentice-Hall, Englewood Cliffs, NJ (1991).
10. H. Li and M. Maresca, "Connection autonomy in SIMD architectures: a VLSI implementation, *J. Parallel Distrib. Comput.* **7**(2), 302–320 (1989).
11. K. Hwang and F. A. Briggs, *Computer Architecture and Parallel Processing,* pp. 779–784, McGraw-Hill, New York (1984).
12. R. Negrini, M. G. Sami, and R. Stefanelli, *Fault-Tolerance through Reconfiguration of VLSI and WSI Arrays,* MIT Press, Cambridge, MA (1989).
13. K. E. Batcher, Design of a massively parallel processor, *IEEE Trans. Comput.* **C-29,** 836–840 (1980).
14. M. C. Pease, The indirect binary n-cube microprocessor array, *IEEE Trans. Comput.* **C-26,** 458–473 (1977).
15. C. L. Seitz, The cosmic cube, *Comm. ACM* **28,** 22–23 (1985).
16. W. D. Hillis, *The Connection Machine,* MIT Press, Cambridge, MA (1987).
17. Q. F. Stout, Hypercubes and pyramids, in *Pyramidal System for Computer Vision* (V. Cantoni and S. Levialdi, eds.), pp. 75–89, Springer-Verlag, Berlin (1986).
18. T. H. Lai and W. White, "Mapping pyramid algorithms into hypercubes, *J. Parallel Distrib. Comput.* **9,** 42–54 (1990).
19. I. Hussien, Pyramid algorithms implementation on the connection machine, in *Image Understanding Workshop,* pp. 634–639 (1988).
20. C. A. Sher and A. Rosenfeld, A pyramid programming environment on the connection machine, *Pattern Recog. Lett.* **11**(4), 241–245 (1990).
21. T. Mzaik and J. M. Jagadeesh, Simulation of SIMD pyramids on the connection machine, *Proc. ISMM Int. Work. Parallel Computing* (D. Marino and G. Mastronardi, eds.), pp. 378–381, Acta Press, Anaheim CA (1991).
22. A. Rougerie and A. Mérigot, Architectural simulation of a fine-grained parallel pyramid computer on the connection machine, Proc. CAMP 91, 1991, pp. 297–307.

Chapter 8

Heterogeneous Hierarchical Systems

It is difficult to match the computational requirements of the various processing steps (low to high) in a vision problem with just a single homogeneous architecture. Hence, some heterogeneous systems composed of at least two main parts have been proposed: performance and flexibility suggest the use of a MIMD system for the high-level stages. For reasons of efficiency a large amount of data requires specialized hardware to process the image at the low-level stages (e.g., standard low-level image processing can be effectively performed by SIMD systems with fine granularity or by pipelined systems). Examples of composite systems which will be described are heterogeneous pyramid systems (such as the Warwick project), multilevel arrangements of meshes (IUA), and reconfigurable multimode parallel machines (PASM and Array/Net).

8.1. INTRODUCTION

In the taxonomy of hierarchical architectures introduced in Chapter 4, heterogeneous systems are split into two families: loosely coupled and closely coupled systems. The distinction refers to the way in which the layers of the systems are interconnected rather than to the interconnection scheme of the processors on each layer.

Closely coupled systems interpose memory banks between the layers that

make up the hierarchy. The synchronization of interlevel data exchanges can be based on a variety of primitives, as with any tightly coupled system. A detailed discussion of the control issues for data exchanges in the hierarchical environment is given in Chapter 9. Loosely coupled systems adopt explicit links (buses or permutation networks). Interactions between the levels usually require a less tight synchronization in this case, especially when the interconnection network works according to a packet-switching mechanism.

With reference to the number of levels into which heterogeneous systems are organized, the proposed and realized hierarchies consist of three levels at the most. The three-level model for computer vision, discussed at length in Chapter 4, is the paradigm on which these hierarchical systems are designed.

The predominantly local, fine-grained computations of the first stages in image processing are supported by a SIMD subsystem that constitutes the first level of the hierarchy. Middle- and high-level vision tasks are handled by coarse-granularity processors in the second and third levels of the systems. While the SIMD subsystem is designed around custom chips, the upper levels are based on commercially available microprocessors.

Actually, the distinction between a second level (dedicated to the middle stage) and a third level (dedicated to the highest stage) is at least as uncertain as the corresponding partition of vision tasks into the three stages. Moreover, some heterogeneous systems in the upper levels of the hierarchy concentrate both coarse-granularity computations as well as control tasks for the fine-granularity subtasks. This highlights a predominant use of the levels in the hierarchy for control strategies rather than for multiresolution representation-based computations.

For this reason, the subsystems that make up the heterogeneous hierarchical machines considered here are rather independent components. It is difficult to identify a pyramid structure within the hierarchy. The ratio between processors of adjacent levels changes when we consider the transition from the first level to the second and from the second to the third. Typically, one processor in the second level interacts with a few dozen small-scale simple processors in the first level, while a processor in the top layer of the hierarchy handles a number of intermediate-level processors in the range of a single dozen.

Yet, we find a residual pyramid organization in the ''cluster'' concept, used in at least one of the systems described. A cluster is a modular subset of a system consisting of a small mesh of chips integrating part of the low-level system; one coarse-grain processor from the upper levels; external memory for each type of processor; and interconnecting circuitry. Physically, the cluster is realized usually as a single board. At the onset, it acts as the first small-scale prototype of the system under construction (it is a minimal hierarchical system

if the control function is carried out by the coarse-grain processor). Once it is shown to be functioning properly, it becomes the module for assembling realistic machines.

Developing algorithms and applications on hierarchical systems is in itself a difficult task, but even more so when the system is heterogeneous. On the system software side, compilers must produce and optimize codes for quite different target machines, with each one being a parallel system. The higher levels of these computational engines are better equipped from this point of view, because many commercial versions of standard high-level languages are available on the market. The (usually SIMD) lowest level of the hierarchy is obviously the least supported. On the programmer side, the programming support for managing the heterogeneous hierarchical system is practically minimal. At best, the perception of the system is that of two or three parallel subsystems that interact through some more or less explicit synchronization mechanisms.

In the following sections, we describe the more relevant systems that have reached a prototype stage (the Array/Net described in Section 8.5 is an exception). Many other heterogeneous hierarchical machines have been proposed and built. We have restricted our discussion to those systems that effectively use the hierarchy for processing purposes, not just for a hierarchical distributed control.

8.2. WARWICK PYRAMID SYSTEM

The heterogeneous pyramid known as the Warwick pyramid machine[1, 2] (WPM) is a three-level architecture, with each level performing computations at a different granularity.

The low-level iconic layer consists of fine-grained bit-serial PEs that make up a rather standard array processor. Instead of designing and fabricating a new chip, AMT's distributed array processor (DAP) circuits[3] have been used.

The intermediate layer consists of a set of intermediate-level processors (ILP in the following), built with off-the-shelf bit-slice components (the AMD 29000 family). The purpose of the ILP is to drive the fine-grained PEs and to collect and aggregate data from the iconic layer.

At the top of the processing hierarchy, we find a network of INMOS transputers that form the so-called symbolic layer, responsible for high-level vision tasks and for dispatching intermediate- and low-level operations to the other levels.

The Warwick pyramid architecture aggregates the three levels in clusters: a cluster consists of a 16×16 subarray of DAP PEs, one ILP for fine-grain control, and one transputer. This small-scale heterogeneous pyramid is the

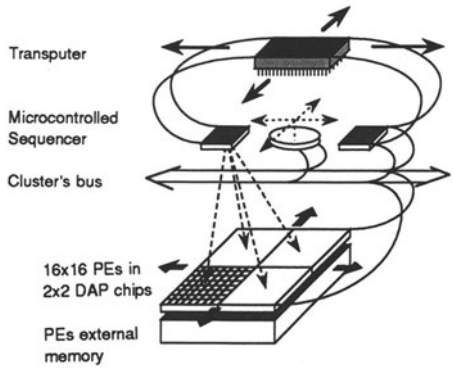


Figure 8.1. The three-level structure of the Warwick cluster. The top level consists of a transputer; the intermediate level hosts the microcontrolled sequencer with its program and data memory; four DAP chips make up the bottom layer for a total of 256 PEs. Vertical transmission is realized by dual-ported memory sharing.

building block for large systems. With proper communication mechanisms at each level, clusters can be interconnected in various topologies (linear, mesh, ring). The resulting system supports the Multi-SIMD operative modality by allowing each cluster to carry out a private task (Multi-SIMD across space). SIMD and SPMD processing is also possible, thanks to synchronization hardware at the ILP level that brings all subarrays under the control of a single source of instructions. The hierarchy in the system allows for more control parallelism, since a transputer is capable of time sharing its resource between more than one process and can thus perform computations and issue control instructions for the ILP concurrently.

8.2.1. Structure of a Cluster

A sketch of the structure of cluster is given in Figure 8.1. The 256 DAP PEs are arranged as a 2×2 mesh of four DAP chips: each such chip hosts an 8×8 submesh of PEs. For a detailed description of the DAP architecture the reader should consult the references. We will only briefly mention the characteristics of AMT chips that are relevant to the present discussion.

8.2.1.1. SIMD Low-Level Processor

A three-input full adder is the very simple processing unit of the DAP PE (see Figure 8.2). Inputs to the adder come from more than one source, including two local Boolean registers, the memory, one of the four near neighbors, and a data bus. The carry output is distributed to the neighbor, and the sum

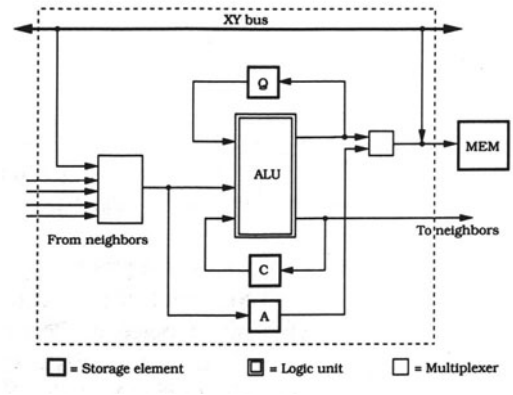


Figure 8.2. Simplified block diagram of the DAP PE. The PE is based on a bit-serial full adder, 4-connectivity with multiplexing for neighbor access, and *x*/*y* addressing capability within the chip.

output is latched in a Boolean register and can be transmitted to the data bus and to memory. Near-neighbor processing is based on the multiplexing technique (for a discussion of the alternatives to this type of processing, see Section 5.2.2.1). The usual operation autonomy is implemented by conditioning instruction execution with one of the Boolean registers. With regard to storage, in the current Warwick prototype each DAP PE has access to 64 kbits of external memory.

The 8×8 mesh of PEs within the AMT DAP chips is also interconnected by eight row and eight column data buses that can be read and written to by the PEs. This facility is used in the Warwick cluster to implement the two associative operations through which the ILP interrogates its "slave" SIMD submesh. These two operations are discussed later.

8.2.1.2. Intermediate-Level Processor

The ILP is composed of a microcontrolled bit-slice pair containing a 16-bit ALU (AMD 29116) and a sequencer (AMD 29331). Besides the microcode store, the ILP has access to a dual-port RAM used to exchange data with the SIMD subarray and with the transputer. With respect to the classification introduced in Chapter 4, the WPM is therefore a closely coupled system. The ILP is driven by horizontal microinstructions that allow the various processing facilities to be controlled in parallel, including the DAP array, so that the ILP and the DAP chips can be seen "as a single processor, capable of executing both scalar and array instructions."[1]

From the point of view of control flow, the sequencer accepts signals from the serial ALU for condition handling and from the DAP subarray data bus to

implement a some/none local test. Indeed, by addressing the row and column buses, the ILP can detect the status of the PEs and build a signal for program flow control. This is the normal global-OR mechanism of SIMD processor arrays (see Section 5.2.3.1), here limited to the dimension of a single pyramid cluster. Furthermore, the bidirectional data buses allow the ILP to directly address a single PE (by properly masking columns and rows). Some/none detection and PE addressability are the first type of associative data collection available to the ILP.

A second form of associative processing is obtained within the cluster with the aid of an adder VLSI circuit interfaced to the DAP external memory lines (an analogous scheme was adopted in the tree-of-adders system of the GAM pyramid; see Section 5.2.4.3 for details). The semicustom VLSI adder produces the count of the PEs in the DAP chip that respond to a condition broadcast by the ILP. The time required for the count to be stable is compatible with the DAP basic cycle time.

8.2.1.3. Symbolic Processor

The INMOS transputer microprocessor is the symbolic processor of the cluster. It acts both as the cluster controller and as a working processor. As controller, its primary purpose is to load (at boot time) the sequencer microcode memory with the intermediate-level program to be carried out. Private access to the microcode memory is reserved to the transputer for this purpose. The downward flow of control reduces the required bandwidth for driving the array of bit-serial PEs by embedding commands for the ILP in macros. The expansions of these macros is performed by the sequencer.

The transputer controls the ILP by allocating a ''process'' to the vertical communications: program downloading and data collection. The second activity is implemented through the dual-port memory shared with the ILP. Other ''processes'' run concurrently on the transputer. Typically, they carry out higher-level tasks that interact with the ILP by sending requests for information available in the image. Since the aggregation of the image in regions is much more irregular than the subdivision of the image into subarrays, the tasks on the transputer interact between themselves and with analogous tasks on other symbolic processors from neighboring clusters.

8.2.2. Clusters Interconnection

The configuration of the Warwick system in terms of clusters can be tailored to the application. Pyramid processing requires a bidimensional arrange-

ment of clusters into a mesh, so the image at the base of the pyramid is subdivided into tiles. However, clusters can also be interconnected in a linear string or in other topologies. We would like to point out that the feasible topologies depend very much on the required low-level connectivity rather than on the interconnections at the symbolic layer.

Indeed, if there is a relevant activity of data exchange at the pixel level, the bidimensional topology of the subarrays imposes a fine-grain synchronization and the setting up of a SIMD modality between clusters. On the contrary, if data once aggregated in the ILP–transputer pair are shared predominantly, the interconnection patterns become more flexible, since the Warwick system can adopt the numerous solutions derived to build a network of transputers.

If we remain in the domain of pyramid processing, a typical situation is the execution of a program that schedules one (or possibly more) tasks on the transputers that control the clusters. The system is synchronized as a whole unit to carry out SIMD operations at the lowest level. To do so, a primitive at the ILP level initiates a rendezvous procedure. Each ILP has a 4-connected switched wired-OR network: this line will be set only when all the sequencers are ready. By selectively enabling the switches of the network at each ILP, it is possible to set up variable configurations of clusters willing to synchronize. The parallel data paths of the bit-serial PEs adapt themselves to the established topology once the synchronization process has identified the set of connected clusters. When the clusters operate in multi-SIMD mode, the edges of the subarrays are connected in torus mode.

8.2.3. Programming Environment

The rich hardware environment of the Warwick system demands different programming capabilities to optimize performance at each level.

The low-level iconic layer is usually programmed according to the "data parallelism" paradigm (a discussion of the approaches to the programming environment for hierarchical systems is presented in Chapter 9). This is typical of SIMD architectures. The flexible, coarse-grained network of transputers of the symbolic layer is more suited to the communicating sequential processes paradigm. No single environment supports both programming styles in a unified manner. Therefore, the first running Warwick prototype is programmed with a variety of tools.

Assembly language is the only means of coding the low-level routines. An instruction in the assembler covers both the actions of the sequencer and the code to be executed by the DAP array. This type of programming is very difficult and requires a deep knowledge of the details of the hardware (a very

similar approach was followed in the design of the system software in the PAPIA 1 prototype; see Chapter 5). The high-level network is programmed with one of the parallel C dialects typical of the transputer environment.

The designers of the Warwick pyramid are working toward a unified programming environment. It should retain the efficiency offered by the knowledge of the hardware for the experienced user, but otherwise it should hide the intricacies of the heterogeneous parallel system from the standard programmer.

The object-oriented paradigm is considered a viable solution.[2] The class abstraction is the foundation for a homogeneous programming style. It defines the data structure and the procedures (''methods'') available on such structures as a unit. Both the data structure and the methods are tailored to the hardware: on the transputer and on the DAP array the same procedure is embedded in different methods, each optimized for the hardware used. The implementation is, however, hidden in the class abstraction, and the user perceives the objects derived as self-contained building blocks. In the hierarchy generated by the inheritance mechanism of the object oriented paradigm, the topmost class is the ''cluster,'' which defines all data structures available in the physical heterogeneous small pyramid, along with the available operations.

The Warwick pyramid machine is an example of parallel platform for experimenting hierarchical processing strategies with a limited-tasks hierarchy. The three-level structure of the system (actually a two-level architecture if we consider the ''true''processing units) allows us to implement a master–slave paradigm rather than a multilevel computational strategy.

8.3. IMAGE UNDERSTANDING ARCHITECTURE

A joint effort of the University of Massachusetts and of Hughes Research Laboratories for over five years led to the design and to the realization of a three-level, hierarchical and heterogeneous parallel architecture, known as image understanding architecture (IUA). While not conceived in the spirit of a multiresolution architecture, the IUA[4, 5] embeds a hierarchy of processing levels aimed at matching the traditional partition of computer vision into three stages: high-level (symbolic), intermediate (associative), and low level (iconic).

8.3.1. Three-Level Hierarchy

On the basis of the requirements of each phase, summarized in Weems,[6] the designers conceived a specialized parallel subsystem for each level. The

vertical connections between the processing units are implemented through banks of shared memory, so this system in the taxonomy quoted was included among loosely coupled systems. The cluster concept of other heterogeneous architecture, such as the WPM in Section 8.2, is not retained explicitly here. However, each processor in the symbolic layer is the apex of a three-level heterogeneous pyramid built with a reduction factor of 64 (see Chapter 3). Thus, the minimum IUA configuration hosts an 8×8 arrangement of intermediate-level processors and 4096 bit-serial PEs in the iconic layer. Instead, the full-scale IUA consists of a 512×512 mesh of bit-serial PEs, of a 64×64 mesh of intermediate processors and of 8×8 symbolic processors. Both the granularity of the processors and their intralevel interconnection modality were chosen to suit the specific requirements of the levels.

The processing modes supported in the IUA cover the wide spectrum from strict SIMD and multiassociative SIMD mode of the low-level content addressable array parallel processor (CAAPP), to SIMD and MIMD mode in the intermediate communication associative processor (ICAP), then to general MIMD mode in the symbolic processing array (SPA) of the highest layer in the architecture. An array control unit (ACU) takes commands from the SPA and manages both the ICAP and the CAAPP. In the following, we shall give a short description of each level, along with a discussion on the control strategy and programming environment.

8.3.1.1. Content Addressable Array Parallel Processor

The low-level iconic processor has been designed with custom-integrated circuits that embed the bit-serial processors, local memory, and a special reconfigurable mesh, called the *coterie network*. The reconfigurable interconnection capability of this mesh allows the PEs to set up, at run time, groups of processors that receive the same instruction but operate on private data common to each group (hence, the ''coterie''). This data-dependent, associative mode constitutes a notable enhancement of SIMD modality: it can be used for many low-level algorithms that are based on a regional paradigm.

The structure of the CAAPP PE consists of four functional subunits (see Figure 8.3) described in the sequel.

- *Registers.* Five single-bit registers form the immediate operands of the PE processing unit. Two of them have special functions. The A register is the activity bit and controls the status of the PE (see Section 5.2.2.4 for a discussion of operation autonomy). The X register stores the ''respond'' bit,

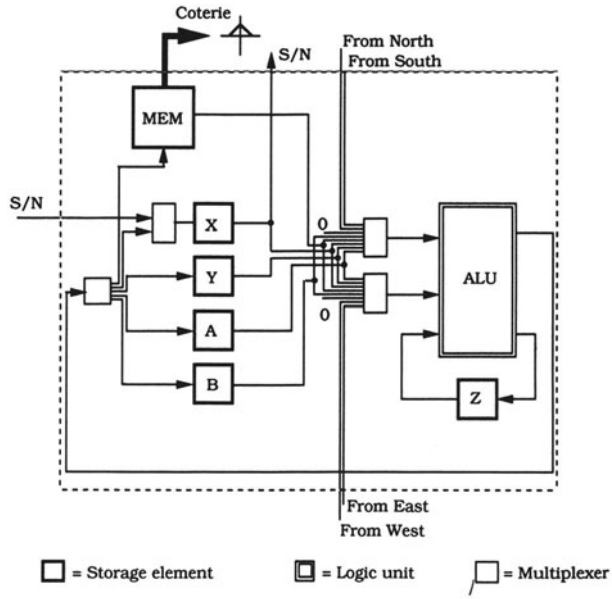


Figure 8.3. Simplified diagram of the CAAPP PE architecture. Note the connectivity autonomy supplied through the direct control of the coterie settings by the local memory.

which drives the two associative computations carried out within the CAAPP: the some/none and the count mechanism. The first is the global OR signal used as a Boolean feedback to the controller of the array, the latter is the adding circuitry (also available in other systems such as the WPM and the GAM pyramid). In the IUA system, the role of the enabling–disabling activity bit and that of the status information are kept separate.

- *Local memory.* It consists of 320 bits organized in banks of 128 bits (two full banks and a half one). The banks are addressed with 3 bits so that the maximum local memory could be raised to 1024 bits. While the PE sees the local memory as a linear address space with bit addressability, two banks of such a memory can be transferred to and from external memory in words of 8 bits. A corner-turning circuitry was implemented at chip level for this data reshaping. Such transfers adopt a cycle-stealing mechanism, so that these two banks of local memory can be considered as a cache memory.
- *Processing unit.* This is a bit-serial ALU with Boolean logic and a full adder.

• *Interconnection circuitry (coterie).* The near-neighbor interconnection is realized with a specialized broadcasting scheme that uses six transmission gates. These gates are attached to the X register and connect a PE with its neighbors in the four cardinal directions, north, south, east, and west, plus the northeast and northwest. A wired-OR bus containing the X registers of neighboring PEs was established by setting the transmission gate control signals of every PE. The status of the bus (which is managed by a precharging scheme) can be sensed at every PE. The group of PEs in a coterie is the result of a local processing in each PE, but can give rise to large regions of electrically interconnected processors. Many such coteries can be established independently.

The coterie supports three modes of inter-PE communication: broadcasting from a source PE to all participants in the group; point-to-point transmission to a single destination within the group, by previous broadcasting of the identification tag of the receiving PE; associative reading of wired OR messages, whereby a single receiver gets the minimum of the multiple data transmitted within the group on the fly.[7] The coterie control register insists on the 8-bit data path from the local memory, thus allowing a local reconfiguration in a single machine cycle.

The first CAAPP prototype was assembled with custom VLSI chips, each containing an 8×8 submesh of PEs. The chip has roughly 130,000 transistors, is fabricated in a 2-μm CMOS technology, and operates with a 10-MHz clock cycle. Beside the 64 PEs, a chip integrates the corner-turning and external video RAM circuitry and more circuitry for the two associative data paths, the some/none and the count. Both are implemented with a tree topology and are capable of producing the corresponding aggregate data, starting from the 64 PEs in a machine clock cycle. The count value can be off-loaded from the chip only serially, due to the tight pinout constraints imposed at the time of the first design of the chip.

The next level of the assembly of the IUA is the daughterboard. This board cuts across the first two levels in the system, since it hosts both a CAAPP chip and one processing unit from the ICAP processor. The daughterboard is therefore a cluster within the system. It is organized around a system bus, on which the dual-ported memories that interconnect the CAAPP and the ICAP are interfaced. Moreover, the CAAPP chip directly interconnects with two video RAMS: the first is for image loading or unloading to or from a VME device, the second is the external working memory for the PEs. A third video RAM interfaces the daughterboard with the symbolic processor. The ICAP

processor accesses its static memory (used both for data and programs) through the daughterboard bus. In the full IUA system, 64 daughterboards would be interconnected to a single motherboard.

The CAAPP low-level processor is being updated according to recent advances in VLSI technology. A 256-PE chip is under fabrication with a 1.25-μm geometry.

8.3.1.2. Intermediate Communication Associative Processor

The second layer in the IUA hierarchy consists of general-purpose 16-bit DSP chips (in the first prototype, Texas Instruments TMS320C25), chosen for their speed in arithmetic operations (at 5 million instructions per second, one such processor executes a multiply-and-accumulate operation in one machine cycle). Sixty-four DSPs, each with 256K of local memory make up the IUA prototype ICAP processor. The next release of the system will replace the 16-bit DSPs with 32-bit ones, 50-Mflop components, and the local memory will be raised to 1 Mbyte.

With respect to the Warwick machine, the designers of the IUA adopted a different approach to fulfill the requirements of intermediate-level vision tasks. The instruction set of the DSP and its arithmetic capabilities were considered very important for geometry-related operations. These operations are typically performed on tokens built from low-level groupings: line segments, shape features, etc. To compensate for the reduced interconnection capabilities of the DSP, a special interconnection network was designed around a custom VLSI chip, called the PARCOS chip (parallel communication switch).

This chip[8] implements a 32×32 crossbar switch with an internal configuration memory for storing privileged interconnection patterns. The switch supports both point-to-point and broadcast communications. The configuration memory can be loaded with new configuration patterns without interfering with chip operations. A new configuration pattern can be activated dynamically with a single control memory write operation. The 64-input, 64-output interconnection network of the ICAP processor links the 5 Mbits/sec serial ports of the DSP with a two-stage network of PARCOS chips. The whole network is managed centrally by the array control unit.

The second version of the IUA system will no longer use the dedicated interconnection network, but will rely on high-speed direct memory channels implemented in the globally shared memory of the ICAP processors.[4]

8.3.1.3. Symbolic Processing Array

The top level of the IUA hierarchy is in charge of symbolic processing. The target system will support artificial intelligence software paradigms, based on the blackboard concept, with a MIMD array of general-purpose serial processors. Connectivity with the lower ICAP will be realized through memory sharing on the video RAM located for this purpose on the daughterboards.

In the design of the system, the highest level of the hierarchy groups the clusters in the daughterboards through a motherboard. A motherboard provides 64 slots for connection with the daughterboards. This high-level master board will contain both the interconnection network of PARCOS chips for ICAP connectivity and a third type of dedicated, special-purpose VLSI component, the ''feedback concentrator'' chip.[9]

The feedback concentrator chip embeds the trees of the two associative data paths originating from the CAAPP chips in the daughterboards. It produces the final global signal some/none with a latency of one CAAPP machine cycle. Thus, the global OR signal is available to the ACU two machine cycles after instruction dispatching to the CAAPP. The feedback concentrator also delivers the global count from the daughterboards' local count with a mixed serial and parallel output. The 64 least significant bits of the local counts are fed to the feedback chip adder inputs, and the first low-order bit of the result is available at the serial output in a single CAAPP cycle. In the meantime, the high-order 6 bits are available at the parallel output. For multibit counts, the successive bits in the operands are fed to the concentrator chip that recirculates the partial result available at the parallel output. Therefore, the final count is available exactly one CAAPP cycle after the most significant bits of the local counts have been fed to the concentrator chip.

The plans for the final SPA layer[4] envisage the use of a commercially available multiprocessor.

8.3.2. Array Control Unit and Programming Environment

The IUA system is specially suited for a hierarchical control strategy. The symbolic layer is a self-supporting general-purpose MIMD multiprocessor that dispatches tasks on the lower layers for iconic related enquiries. The expected response time (around 10 μsec) of the SPA processor to the feedback from the lower layers is too long to allow direct control of the system from this level. Furthermore, the sustained speed of the CAAPP level and the fine granularity of the instructions of the bit-serial PEs require a dedicated controller.

The array control unit is the subsystem that interfaces the SPA to the ICAP

and CAAPP as far as control is concerned. The tight connections between the layers of the IUA realized through dual-ported memories are used essentially for massive data exchanges rather than program downloading. However, a shared memory is used for transmission of tasks and for control between the SPA and the ACU.

The ACU uses the VME bus to interface with the daughterboards, mainly to store the programs in the ICAP DSP RAMs and to initiate operations. At this level, the ACU is able to keep the ICAP under control through a standard, dedicated microprocessor (the plans are for a 68020 chip). Feeding the CAAPP is a much more demanding operation, and the ACU is therefore split into a macrocontroller and a microcontroller (a popular solution on most SIMD arrays).

Control flow at the SIMD–SPMD level is managed using the some/none response from the motherboard and the count data path. The setup of the configuration of the DSPs network is also an ACU task.

As in all heterogeneous systems, setting up a productive environment for program development and applications execution is a very complex task. The first running prototype of the IUA system offers high-level extensions to C and to FORTH to program the CAAPP. Admittedly, such extensions require deep knowledge of the machine hardware.

The designers of IUA aim at building a programming environment that hides the variety of processors and topologies of the system as much as possible. The targeted situation is one in which the programmer perceives the system as an augmented von Neumann machine with a massive intelligent memory. The goal is a single unifying model and language for programming the IUA so that programmers will not have to distinguish explicitly among the three levels.[4] As in the Warwick pyramid case, the object-oriented approach is the best option. It is not certain whether the object paradigm will be used at all levels of the machine or if some specialized communication libraries will be used to code message exchanges at the ICAP level.

8.4. THE PASM

The partitionable SIMD/MIMD system for image processing and pattern recognition (PASM) is a long-lasting design effort carried out by Siegel *et al.*[10, 11] to explore the benefits of a reconfigurable hierarchical structure, particularly suited to supporting both synchronously orchestrated (SIMD) and independent (MIMD) computations.

8.4.1 Partitionable Two-Level Hierarchy

An overall description of the system allows us to recognize two major processing subsystems, arranged in a two-level hierarchy: the parallel computation unit (PCU) and the microcontrollers unit (MC). If we include the external host driving the microcontrollers unit, we end up with a three-level system, as depicted in Figure 4.4. As discussed in Chapter 4, the heterogeneous hierarchy embedded in PASM can be ascribed to the loosely coupled family. Indeed, the PCU is linked by dedicated buses to the MCs, and the association between processors and microcontrollers is static. The PASM system is, however, further enriched by a secondary storage subsystem that can be accessed both from the PCU and from the MC, so its classification as a loosely coupled system is somewhat inappropriate.

The major novelty in the PASM system is the concept of system partition. The PCU is designed to contain $N = 2^n$ processors with a capacity to be reconfigured as $Q = 2^q$ independent subunits under the control of Q proper subsets of microcontrollers. Each MC thus has to manage at the most N/Q slave processors in the PCU. In such a case, the system works according to the SIMD or to the Multi-SIMD paradigm. Furthermore, owing to the coarse granularity of the processors that make up the PCU, we can even reconfigure the system to operate as a single MIMD machine. Finally, a mixed-mode configuration is possible as well: the machine is split into two or more submachines, and each submachine can run in SIMD or MIMD mode independently of the others. Thus, the PASM architecture is actually a test-bed for implementing a rich variety of multiprocessing modes, from strict SIMD to SPMD, from Multi-SIMD to MIMD.

The PASM system was designed for a maximum configuration of $N = 1024$ processors in the PCU and $Q = 32$ processors in the MC unit. The actual running prototype is a scaled-down version of the full system: it has 30 processors all in all. In the following we will concentrate on the processing subsystems that make up the two-level hierarchy in PASM.

8.4.1.1. Parallel Computation Unit

The low layer in the PASM hierarchy consists of $N = 2^n$ processor–memory couples that support the computations. Each processor can either be a specialized component, specially suited to low-level image tasks, or a coarse-grain microprocessor (the running prototype uses MC68000 microprocessors locked at 8 MHz); it will be referred to in the following as the processing unit (PU). It has access to a local memory, which is used for data and program storage.

The local memory is logically split into two banks; this organization allows the double-buffering technique for data exchanges with secondary storage.

The major component of the PCU is the interconnection network that links the PUs. Unlike other heterogeneous systems thus far considered, PASM adopts a flexible approach to PU interconnection and does not commit itself to a single topology. The interconnection network is a multistage subsystem based on the circuit-switching approach. Siegel investigated the use of different schemes[12] to design a reconfigurable network that suits the PASM requirement of system partitionability.

A possible choice is a generalized cube network, which consists of $n = \log_2 N$ stages: each stage has N inputs and N outputs, and the rearrangement of paths between the inputs and the outputs is executed by $N/2$ two-input two-output interchange boxes. The boxes support four configurations: pass through, exchange, lower broadcast (lower input is connected to both outputs, upper input is discarded), and upper broadcast. One-to-one communications require n-bit address tags to set up the circuit in the network: stage j in the network processes the jth bit position in the tag. Longer tags are necessary to realize broadcast-type communications.

Such a network can be partitioned into Q subnetworks, which are independent of one another, to support the subdivision of the PCU into more subunits. The routing mechanism within each subnetwork follows the above-quoted algorithm, with minor modification needed to comply with the separation. Actually, the PASM prototype embeds a fault-tolerant variation of this network, known as an extra stage network.[12]

The PCU can operate in SIMD, Multi-SIMD, or MIMD mode. In the second mode, each partition executes the same program distributed by the microcontroller(s) that manages the partition. Both in SIMD and in Multi-SIMD mode the system has to use the multistage interconnection network even for short near-neighbor data exchanges. The flexibility introduced by the network is better exploited in MIMD general-purpose communications than in the tightly orchestrated ones on SIMD mode.

The granularity of the PUs used in the prototype allows us to speculate that a full-fledged PASM system operates in SPMD mode, rather than in SIMD mode. The synchronization of a set of standard microprocessors is a quite different task compared with the analogous synchronization of the *custom-made* PEs of typical SIMD-only array processors. However, the collection of status information for data-dependent condition handling is much more complex, since standard microprocessors do not have a status register accessible from the outside. The solution of such problems was one of the significant results of the PASM design effort.

To this purpose, the prototype PASM embeds support circuitry at every PU for interfacing with the MC.[13] This additional hardware consists of an instruction broadcast unit (IBU) and a condition code logic (CCL). The IBU activates the dispatching of SIMD instructions from the MC when a PU executes a memory fetch from a special area of its local address space; the actual dispatching is carried out only when all the PUs that are enabled for this instruction have executed the fetch from memory operation (barrier synchronization). The CCL conveys status information from the PUs to the MC. Although dedicated hardware speeds up the generation of the test in each PU, the protocol for the overall transfer is rather long. When measured it was seen to be a major source of overhead in the running time of conditional intensive algorithms.[13] The situation in which more MCs belong to the same partition is even more complex. It will be briefly considered in the next section.

8.4.1.2. Hierarchy of Microcontrollers

The second layer of the PASM is made up using Q independent microcontrollers. Each microcontroller is itself a processor with local memory (a double-buffering scheme is also adopted with the microcontrollers to ease the exchanges with the host computer) and manages a fixed number (N/Q) of PUs. The PASM prototype uses MC68000 microprocessors (we note that with this instantiation the PASM is no longer a heterogeneous machine, strictly speaking).

The setting up of partitions in the system is constrained by the power-of-2 subdivision but is otherwise free: any number R ($1 \leq R \leq Q$) of partitions, or virtual machines, can be created by loading the memories of the MCs with replicated copies of the R programs. To support data sharing among microcontrollers of a partition, a special reconfigurable bus was proposed. Microcontrollers and memory modules have access to the bus, which can be split into more independent units by special reconfiguration switches. It is thus possible to create electrically isolated regions of microcontrollers and memory modules with each region being associated with the corresponding PASM partition.

Since partitions can contain different numbers of PUs, two types of addresses exist for individuating a PU: a logical address, unique within the partition, and a physical address, unique for the whole system. The mapping of the two addresses is done at compile time, when the programs specify the configuration of the system they want to work with.

Independently of the reconfiguration bus, PASM interconnects the microcontrollers with a communication bus used for signaling and synchronization.[14] Indeed, when the operative mode is SIMD (or even Multi-SIMD), aggregate

operations and global tests are executed in the PCU to produce status information for the microcontrollers. Such status information is used for program flow control (the global-OR mechanism described in Section 5.2.3.1). Owing to the partition architecture of PASM, the generation of a single status signal valid for all microcontrollers within a partition requires a fairly complex protocol on the communication bus. Briefly, the microcontroller that initiates the exchange broadcasts the identification of the task and the local status value; all other microcontrollers that contribute to the same task write their local status information on the communication bus data line in a wired-AND modality. This information can eventually be sensed from all participating microcontrollers. This protocol is handled completely from the MC layer. The host computer has no part in it, not even in a totally SIMD system configuration.

8.5. ARRAY/NET PROJECT

Another type of reconfigurable hierarchical system was proposed long ago by Uhr. The Array/Net[15] consists of a two-layered parallel machine designed to experiment with various forms of reconfiguration, among which is processor word size.

Basically, the proposed Array/Net system consists of a huge number of simple processors, each with a small-width data path. When used for higher-level computations, such processors are grouped to form fewer processors with a larger word length. This two-level hierarchy can be extended. Conceptually, the process can be brought to an extreme by starting with an array of bit-serial PEs, configured as a standard SIMD array processor. A first higher level is then obtained by configuring each set of N PEs as a single N-bit coarse PU; a second level consists of N^2-bit PUs in a number equal to $1/N$ of the PUs in the second level; and so on. The result is a converging pyramid of ever more powerful processors.

Even though only one layer of such a pyramid is active at a time, the increased granularity introduces more flexibility with regard to the processing mode. From a strict SIMD operation for the fine-grained PEs, the system can move to Multi-SIMD or even MIMD mode when operating at a coarser granularity. This second type of reconfiguration, very similar in concept to that of PASM, requires a flexible network for communications among PUs. As the system name suggests, the basic topology supported in all system configurations is the array, which is a two-dimensional mesh.

Although the Array/Net never became a working prototype (the designers

maintain that the construction of the prototype was beyond what reasonably could be achieved with the $150,000 total budget available for the project in 1979), the system was carefully designed up to board level. While the solutions chosen are today out of date on the technological side, the underlying motivations are still valid and have found a partial (though independent) realization in other machines. In the following, we give a very brief summary of the architecture proposed.

The raw processing power of the Array/Net comes from a set of 512 4-bit, bit-slice chips from the 2900 family. A pair of such chips (plus 16 kbytes of local memory) makes up an 8-bit slave processor.

The first configuration of the system uses the slave processors in groups of 16. Each group is laid out as a 4×4 mesh: near-neighbor interconnectivity is realized through *ad hoc* registers. The local memories are used exclusively by the PUs as far as write operations are concerned, but are otherwise shared according to mesh topology. Each 4×4 mesh is driven by a "master" controller that cycles through 64-bit microcode words. The controller memory is mapped in the slave local memory banks. The resulting 16 groups of master and slave processors are themselves arranged in a 4×4 mesh on a single double-Eurocard board. The overall system therefore consists of a midsize mesh of 16×16 8-bit processors, operating in SIMD mode (obviously, 16 copies of the same program must be loaded in the memory of the master processors). A Multi-SIMD mode is also possible and only requires a partition of the system into disjoint sets of boards with each set operating in SIMD mode.

The 16 boards of the systems can be configured as a single MIMD machine in which the master processors have the role of the PUs. The bit-slice approach to board design allows Array/Net to configure the master processors as 8-bit, 16-bit, or 32-bit processors. Actually, a master processor is made up of one or more slave processors, with total control over the 256 kbytes of local memory in its board. The PUs interconnection is primarily obtained through memory sharing at the borders of the groups of slave processors. Furthermore, a bus interconnects the masters between themselves and with an external host computer. The preferred processing mode for this system configuration is MIMD, since master processors have the granularity to support complex computations.

The targeted applications for the Array/Net cover both image processing and numerical intensive array processing operations. The lack of a parallel access to the neighbors, when the system operates in SIMD mode, is a definite drawback in low-level vision tasks. The 8-bit architecture of the PE–PU was considered more effective for the overall performance of the machine.

8.6. CONCLUSIONS

The four systems reviewed in this chapter present a good spectrum of the possible architectural configurations of heterogeneous hierarchical systems.

As with the family of compact, fine-grained pyramids covered in Chapter 5, the running prototypes are quite underscaled versions of the systems designed. They can be considered "proof-of-concept" experimental machines that bring into being some of the most common concepts regarding the match between an architecture and the granularity of the targeted computation.

The partition of such systems into closely coupled versus loosely coupled is represented in the prototypes on an equal basis. Interconnecting machine layers through shared memories seems more convenient (at least from the economic point of view) than building specialized networks. But this pattern is not a general one if we consider general-purpose MIMD commercial systems.

The programming environment for such heterogeneous systems is definitely a topic that requires extensive research. It is significant that one of the benchmarks actually run on these machines (PASM experiments on sorting[14]) was coded completely in assembly language.

REFERENCES

1. G. R. Nudd, T. J. Atherton, R. M. Howard, S. C. Clippingdale, N. F. Francis, D. J. Kerbyson, R. A. Packwood, G. J. Vaudin, and D. W Walton, WPW: A multiple-SIMD architecture for image processing, *Proc. 3rd IEE Conf. Image Processing and Its Applications,* Warwick, 1989.
2. G. R. Nudd, D. J. Kerbyson, T. J. Atherton, N. D. Francis, R. A. Packwood, and G. J. B. Vaudin, A massively parallel heterogeneous VLSI architecture for MSIMD processing, in *Algorithms and Parallel VLSI Architectures* (F. Deprettere and A. Van der Veen, eds.), pp. 463–472, Elsevier, Amsterdam (1991).
3. D. J. Hunt, The AMT DAP—a processor array in a workstation environment, *Comput. Syst. Sci. Eng.* **2**(4), 107–114 (1989).
4. C. C. Weems, E. M. Riseman, and A. R. Hanson, Image understanding architecture: exploiting potential parallelism in machine vision, *Computer* **25**(2), 65–68 (1992).
5. C. C. Weems, A. R. Hanson, E. M. Riseman, D. Rana, D. B. Shu, and J. G. Nash, An overview of architecture research for image understanding at the University of Massachusetts, Proc. IEEE 10th Int. Conf. Pattern Recognition, Vol II, Atlantic City, NJ, 1990, pp. 379–384.
6. C. C. Weems, Architectural requirements of image understanding with respect to parallel processing, *IEEE Proc.* **79**(4), 537–547 (1991).
7. D. B. Shu, G. Nash, and C. Weems, Image understanding architecture and applications, in *Advances in Machine Vision* (J. L. C. Sanz, ed.) pp. 296–355, Springer-Verlag, Berlin and Heidelberg (1989).

8 D Rana and C C Weems, The ICAP parallel processor communication switch, Proc IEEE Int Symp Circuits and Systems, Portland, OR, 1989, pp 126–129
9 D Rana and C C Weems, The IUA feedback concentrator, Proc IEEE 10th Int Conf Pattern Recognition, Vol II, Atlantic City, NJ, 1990, pp 540–544
10 H J Siegel, L J Siegel, F C Kemmerer, P T Mueller, H E Smalley, and S D Smith, PASM a partitionable SIMD/MIMD system for image processing and pattern recognition, *IEEE Trans Comput* **C-30**(12), 934–947 (1981)
11 H J Siegel, J B Armstrong, and D W Watson, Mapping computer-vision-related tasks onto reconfigurable parallel-processing systems, *Computer* **25**(2), 54–63 (1992)
12 H J Siegel, *Interconnection Networks for Large-Scale Parallel Processing Theory and Case Studies*, McGraw-Hill, New York (1990)
13 S A Fineberg, T L Casavant, and H J Siegel, Experimental analysis of a mixed-mode parallel architecture using bitonic sequence sorting, *J Parallel Distrib Comput* **11**(3), 239–251 (1991)
14 T Schwederski, W G Nation, H J Siegel, and D G Meyer, Design and Implementation of the PASM prototype control hierarchy, Proc Second Int Supercomputing Conf , International Supercomputing Institute, St Petersburg, FL, 1987, pp 418–427
15 L Uhr, J Lackey, and L Thompson, A 2-layered SIMD/MIMD parallel pyramidal 'array/net', Proc Workshop on Computer Architectures for Pattern Analysis and Image Data Base Management, 1981, pp 209–216

Chapter 9

Programming a Hierarchical Structure

This chapter reviews the programming tools that have been designed to work on hierarchical architectures. The topic of a high-level language for programming pyramids is central to the discussion. While the data-parallel model typical of massively fine grained architectures can be used with the pyramid as with any other topology, the hierarchy introduces other issues. As might be clearly seen, programming can be a complex task in these systems, since at least concurrence among different levels of the pyramid (the multi-SIMD model) has to be taken into account. A few examples of pyramidal languages will be described, with particular attention to the handling of parallel constructs. The chapter closes with a discussion of the controlling environment.

9.1. LANGUAGES: AN INTRODUCTION

Programming a parallel hierarchical structure has a number of similarities to programming massively parallel systems, but it is otherwise a much more difficult task. Indeed, the management of concurrence among levels (in multi-SIMD systems) or the handling of heterogeneous environments in machines composed of different processors according to the level introduces complex issues that touch both the programming model and the controlling environment. In this chapter, both issues will be discussed, though with different depths.

The programming model in the hierarchical domain inherits much from

parallel languages designed for generic massively parallel architectures. Since the construction of the first parallel systems (such as array processors of the CLIP family, the MPP, DAP, etc.), the language issue has been tackled first in a pragmatic way, by building a macrolevel assembly language capable of managing the intricacies of the target architecture. However, the designers of such systems have immediately perceived the stringent need for a truly high-level language, offering parallel constructs both for variable declaration–allocation and for program flow control. The available commercial systems are equipped with advanced programming environments, mainly based on standard high-level languages such as Fortran, C, C++, and Lisp.

The programming model that dominates by far is "data parallelism"[1] as opposed to "task parallelism." Algorithms are expressed as a sequence of manipulations on data aggregates; each piece of information in the aggregate is processed with the same operation. This type of processing maps very well on fine-grained massively parallel architectures. Yet it is by no means limited to parallel systems. Indeed, APL[2] and later dialects are constructed exactly to express operations on arrays as primitives of the language and have been implemented mainly on serial computers. Hierarchical coarse-grained systems address more closely the control environment necessary to the implementation of task parallelism. The handling of parallelism usually adheres to the communicating sequential processes (CSP)[3] model and uses standard concurrent languages.

The object-oriented paradigm is also suited to coarse-grained parallelism, with a special emphasis on heterogeneous systems. The mechanism of classes, inheritance, and polymorphism can be used to structure uniformly at a high level the tasks running on quite different processors. The details of the implementation of the "methods" are hidden to the programmer and can be optimized to exploit at the best the processors.

The expression of parallelism within the programming model considered can be approached in different ways. A first issue is the visibility of the topology of the target system. A second problem is the semantics of control flow statements, if any at all. We briefly discuss these items to set a common ground for the analysis of the high-level languages proposed and/or built for fine-grained pyramid systems.

9.1.1. Collection-Oriented Languages versus Processor-Oriented Languages

Languages supporting data parallelism can be split into two classes: *collection-oriented,* according to the definitions of Sipelstein and Blelloch,[4] and *PE*

oriented, following somewhat the motivations of the designer of PYR-E.[5] The discriminating features are the visibility of the topology of the parallel systems and the availability of dedicated parallel constructs for controlling the flow of the program; however, they do not create a true partition, because such definitions cannot be stated on a formal basis.

In collection-oriented languages, the data aggregate, called a *collection,* is a language primitive representing the atomic unit of data; the other class of primitives is composed of the *operations,* which take the collections as operands. The use of iterative constructs, such as loops, to act on the collections by examining each constituent element is explicitly banned from the language. With this rough characterization, arrays declared in the ''parallel array type'' construct of Parallel Pascal,[6] parallel variables declared within the ''shape'' format of C*,[7] arrays in Fortran 90, and paralations in Paralation Lisp,[8] are all examples of collections. A more detailed analysis of this concept allows us to distinguish among simple and nested collections (the latter allowing an element to be a collection itself), homogeneous and nonhomogeneous collections, and even ordered versus unordered collections (instances of unordered collections are sets, while arrays with element indexing introduce an ordering among the elements of the collection).

Operations on collections are split into ''apply-to-each'' and ''aggregate'' functions. The first class transforms the collection on an element-per-element basis, using the chosen function consistently on each element; the output is another collection. The second class takes on the whole collection as operand and produces either another collection with a different structure or atomic scalar values. Well-known examples of aggregate functions are set operations, permutations, reductions, and scans.[9]

In collection-oriented languages all syntactic constructs are based on manipulation of collections. When implemented on actual parallel systems, such languages constrain the structures of the collections to resemble the underlying topology. The programmer perceives the topology of the parallel machine only through the data structures by which algorithms are coded.

A typical situation is the synchronous transmission of data from each processor to one of its neighbors. In terms of collection manipulation, this task involves a permutation; when the collection takes the form of an array, the permutation, usually called *shift,* affects one of the dimensions of the array.

A more irregular data movement routine, such as the random access write (RAW) defined for PRAM,[10] hides even more the underlining topology. Basically, this primitive considers a set of sending PEs and a set of receiving PEs: each sender forms a packet containing the data and the address of the

target PE. The intended result of this ''generalized permutation'' is the delivery to each target PE of the data addressed to it, with a mechanism to solve collisions. In terms of collections, such a primitive uses two instances of collections, the data and the indexes (address); the strategy that implements this primitive changes on different parallel architectures according to the underlining topology, but it is (almost) completely transparent to the programmer.

The second class of languages, *processor oriented,* takes the opposite point of view.[5] They express algorithms as explicit representations of the activities of the PEs, including both the management of local resources (registers, memory, etc.) and of the data exchanges through a known and visible interconnection network. For example, synchronous data transmissions are specified by naming explicitly one of the dimensions that make up the neighborhood of each PE within a special neighbor access routine: in a system that supports a multidimensional grid topology, there is a ''shift along grid dimension'' function at least for each dimension physically mapped in the parallel architecture. In general, irregular data movement routines, such as RAW, are not primitives in such languages because they are decomposed in the elementary lower-level functions that ''navigate'' through the topology.

Since the programmer is not constrained to the data structures of collection-oriented languages, PE-oriented languages are intrinsically more flexible and allow one to write programs based on communication topologies other than the physically implemented ones. This is much more difficult when the expression of PE interconnection is concealed in the structure of the collections and of the associated permutations.

Such increased flexibility has a cost. Indeed, it cannot be denied that programs coded in PE-oriented languages tend to be at a ''lower level'' than their counterparts written in collection-oriented languages. According to Mehat,[5] representative languages in this class are PYR-E (described in Section 9.1.5) and CM-Lisp by Thinking Machine Corporation. HCL C (described in Section 9.1.4) can also be included.

Variable declaration is a main issue in both types of languages. Within collection-oriented languages, an association with the parallel environment is implicit in the definition of variables as collections. Standard data types are used to characterize collection elements; moreover, variables declared with the usual typing mechanism are implicitly associated with the controlling serial environment. An explicit declaration of the environment (serial or parallel) at variable declaration is required in PE-oriented languages, since data aggregates are not supported as such.

9.1.2. Semantics of Parallel Constructs

The programming style in the two classes just described tends to be somewhat different, mainly because of the visibility of the target architecture. Yet, the data-parallel programming modality sets a common ground to both language types, when we consider the extensions to the languages required to give a proper meaning to the control flow constructs.

A basic assumption of the data-parallel model is that the transformations on data elements (i.e., the activities of PEs on local data) are uniform (i.e., are specified by a single source of instructions). From the point of view of system architecture, the language has to support SIMD/multi-SIMD [or at most Single Program Multiple Data (SPMD)] multiprocessors modalities. The controller is the only source of instructions and the only actor capable of managing control flow in the program. With the definitions introduced in Section 5.2.2.4, this assumption does not allow for operation autonomy. The control flow structures of parallel languages must therefore be defined so as to specify unambiguously the interactions among the processors and the controller(s).

Parallel languages, based on standard Von Neumann languages such as Pascal, C, and so on, come with only a subset of such constructs. The reason for this limited implementation is the difficulty of creating a clear semantics for each of them. In what follows, we summarize one of the few works[11] which specifically addresses this issue. The approach is formal and is proposed here as a reference. More pragmatic approaches have been followed in the design of the languages of the pyramid systems discussed in later sections.

Levaire and Bougé[11] introduce a set of very simple languages, named $\mathcal{L}$, to be used in the SIMD model. The targeted systems consist of a single controller, a set of processing elements which exchange data through near-neighbor interconnections (the topology of such a network is irrelevant to the definition of the languages), a partial implementation of operation autonomy, which creates the *context* (i.e., the set of PEs enabled to execute the current instruction), and the *global* OR feature, which conveys to the controller a status signal from the whole set of PEs.

The first language includes six basic primitives, namely three instructions (the empty instruction **skip**, the assignment operator **:=**, a near-neighbor instruction **shift**), and three control structures (the sequence **;**, the iteration **while...do...end**, and the conditional execution **where...do...end**). The notation of the language is the following: variables beginning with a capital letter such as X are parallel—that is, they indicate an aggregate that is instantiated in every processor; lowercase variables are serial and exist only within the con-

troller; $X \mid_u$ indicates the instantiation of X on the processor whose address is u; the parallel constants Tt and Ff stand for the logical values true and false when instantiated on every processor.

The semantics of the language $\mathcal{L}$ is constructed along the guidelines of the structured operational semantics.[12] It consists of a set of transitions (indicated by $\rightarrow$) between states; each state is a triplet $\langle P, \sigma, s\rangle$, where P is the program to be executed ($\bigcirc$ denotes the empty program), σ is the environment that associates values with variables, and s is a stack of contexts—that is, a parallel aggregate whose elements only take Boolean values which represent the activity status of the individual PEs. The stack can be manipulated only through the usual Push(s) and Pop(s) operations. Moreover Top(s) reads the top of the stack without modifying the stack. The evaluation of the instruction I in the environment σ is denoted by $[I]\ \sigma$.

Let u be the address of the generic PE. The formal description of the language is the following.

- *Definition of* **context**

$$\text{enabled}(u) \leftrightarrow (\text{Top}(s)) \mid_u = \text{true} \tag{9.1}$$

With this definition, the top of the stack defines the current set of enabled PEs.

- *Definition of* **skip**

$$\langle \text{skip}, \sigma, s \rightarrow \langle \bigcirc, \sigma, s\rangle \tag{9.2}$$

Skip is the no-op code and the transition associated with its execution leaves unchanged both the environment (variables) and the stack (context).

- *Definition of the assignment* **:=**

$$\langle X := Y, \sigma, s\rangle \rightarrow \langle \bigcirc, \sigma', s\rangle \tag{9.3}$$

where

$$\begin{aligned} &\sigma'(X) \mid_u = ([Y]\ \sigma) \mid_u \qquad \text{if enabled}(u) \\ &\sigma'(Z) \mid_u = \sigma(Z) \mid_u \qquad \text{if } Z \neq X \text{ or not}(\text{enabled}(u)) \end{aligned} \tag{9.4}$$

The assignment modifies the environment σ according to the operation Y on enabled processors only. The second clause specifies that the only variable

affected is the one on the left-hand side of the assignment. No side effects on the other variables are possible. The stack remains unchanged.

- *Definition of the near-neighbor primitive* **shift**

$$\langle \text{shift X along d}, \sigma, \text{s}\rangle \rightarrow \langle \bigcirc, \sigma', \text{s}\rangle \tag{9.5}$$

where

$$\begin{array}{ll} \sigma'(\text{X}) \mid_{\text{u}} = \sigma(\text{X}) \mid_{\text{d}^{\circ}(\text{u})} & \text{if enabled(u)} \\ \sigma'(\text{X}) \mid_{\text{u}} = \sigma(\text{X}) \mid_{\text{u}} & \text{if not(enabled(u))} \\ \sigma'(\text{Z}) \mid_{\text{u}} = \sigma(\text{Z}) \mid_{\text{u}} & \text{if Z} \neq \text{X} \end{array} \tag{9.6}$$

The first clause specifies that the source of the data is the processor at location $\text{d}^{\circ}(\text{u})$—that is, in the opposite direction with respect to d. The source can be disabled, because the semantics only requires the destination to be active. This is indeed a "read form neighbor" operation. The stack remains unchanged.

- *Definition of sequence* **;**

Given the rule

$$\langle \text{P}, \sigma, \text{s}\rangle \rightarrow \langle \text{P}', \sigma', \text{s}'\rangle \mid \langle \bigcirc, \sigma', \text{s}'\rangle$$

then (9.7)

$$\langle \text{P; Q}, \sigma, \text{s}\rangle \rightarrow \langle \text{P}'; \text{Q}, \sigma', \text{s}'\rangle \mid \langle \text{Q}, \sigma', \text{s}'\rangle$$

The symbol | indicates an alternative. The first rule indicates that a program P can be transformed into a new one, P′, or finished (empty program ○). The definition of the sequence P; Q stated in the second rule is self-explanatory.

- *Definition of iteration* **while...do...end**

There are two situations:

$$(\exists\ \text{u}\ \{([\text{C}]\ \sigma) \mid_{\text{u}} \text{and enabled(u)}\}) = \text{true} \tag{9.8}$$

$$\langle \text{while C do P end}, \sigma, \text{s}\rangle \rightarrow \langle \text{P; while C do P end}, \sigma, \text{s}\rangle$$

$$(\exists\ \text{u}\ \{([\text{C}]\ \sigma) \mid_{\text{u}} \text{and enabled(u)}\}) = \text{false} \tag{9.9}$$

$$\langle \text{while C do P end}, \sigma, \text{s}\rangle \rightarrow \langle \bigcirc, \sigma, \text{s}\rangle$$

The first clause declares that the controlling condition C is evaluated and, if found true in at least one active processor, causes the execution of the con-

trolled block P. Otherwise (second case), the block P is replaced with the empty program ○; that is, the block is removed. The predicate ($\exists$ u {([C] σ | u and enabled(u) }) is the formal definition of the global OR mechanism (see Section 5.2.3.1). We stress that the context stack is completely unaffected by this construct. The restriction of context, if any, is operated by the executed block P, as well as the modification of the environment σ which affects the evaluation of the terminating condition C.

- *Definition of conditional execution* **where...do...end**

$$\langle \text{where C do P end}, \sigma, s\rangle \rightarrow \langle \text{begin P end}, \sigma, s'\rangle \qquad (9.10)$$
$$s' = \text{Push}(\text{Top}(s) \text{ and } ([C]\ \sigma), s)$$

The conditional execution of a block P according to the parallel Boolean variable C restricts the context Top(s) evaluating [C] σ and modifies the stack s accordingly. The begin...end construct is used to formalize the effect of removing block P. Given the rule

$$\langle P, \sigma, s\rangle \rightarrow \langle P', \sigma', s'\rangle$$

then

$$\langle \text{begin P end}, \sigma, s\rangle \rightarrow \langle \text{begin P' end}, \sigma', s'\rangle \qquad (9.11)$$
$$\langle \text{begin } \bigcirc \text{ end}, \sigma, s\rangle \rightarrow \langle \bigcirc, \sigma, \text{Pop}(s)\rangle$$

The second clause specifies that on removing the block P (that is, when block P is transformed into the empty program), the system restores the context active prior to entering the where construct.

The novelty in the semantics of this very simple language is the normalization of the parallel constructs in terms of manipulation of a single parallel data structure, the stack of contexts. The language could be extended with other more complex constructs. But the major contribution of this operational definition is in the treatment of the exit statements, which modify the flow of control from within block constructs.

Levaire and Bougé augment the language to include an ''escape'' statement that generalizes the *break* and *continue* statements of serial languages. We remember that the *break* statement causes the flow of instructions to exit immediately the current block, with no regard for the controlling condition. The *continue* statement instead causes a jump to the end of the block, with the subsequent reevaluation of the controlling condition.

The extended language $\mathcal{L}^e$ uses the following definition for the **escape** statement.

- *Definition of* **escape**

$$\begin{aligned} &\langle \text{escape}, \sigma, \text{s}\rangle \rightarrow \langle \bigcirc, \sigma, \text{s}'\rangle \\ &\quad \text{s}' = \text{Push}(\text{Ff}, \text{Pop}(\text{s})) \end{aligned} \tag{9.12}$$

The meaning of an escape is to disable all the currently active processors (the stack is popped once and subsequently loaded with the parallel constant Ff false). Since the escape is executed within a block, on exiting the block (which happens when the empty program is executed), the system pops (see Eq. (9.11)) the false parallel constant Ff and thus activates the context immediately external to the block.

The **escape** statements affect only the top of the stack. Accordingly, no exit from nested block constructs is possible, because such an exit would require handling layers located deeper in the stack. The following program fragment is ambiguous, since it is not clear which block the **escape** statement is supposed to exit, either the deeper only, or both.

```
where B1 do
      P1;
      where B2 do
                  P2;
                  escape
                  P3;
      end
end
```

To solve the ambiguity, Levaire and Bougé introduce labeled blocks and labeled escape statements, $\textbf{where}_i$ and $\textbf{escape}_i$. The semantics of the resulting language are constructed on a modified stack of contexts. Each entry in the stack contains the context, along with the label of the closest enclosing block. The semantics of $\textbf{escape}_i$ consists of the recursive analysis of the stack. During this analysis, all contexts created after the first context with label **i** are popped, including the **i**. This restores the context of the block immediately external to block **i**.

Beside being a formal definition of the semantics of parallel constructs, the analysis of Levaire and Bougé can also lead to efficient extensions of existing parallel languages, whenever such languages lack some of the parallel constructs. In Ref. 11 some examples are given in the C* language. In the

following, the high-level languages of the pyramidal systems so far built or proposed are described with reference to the $\mathcal{L}$ language parallel constructs.

A different formal treatment has been introduced by Clermont *et al.*[13] This approach consists of the translation of a parallel program into a program containing only serial control statements and instructions to handle parallel Boolean variables which are the equivalent of the context herewith introduced. A few examples of the approach are given in Section 9.1.5 in the description of the PYR-E language.

9.1.3. HCL, a Pyramid Algebra

The notion of a pyramid algebra is one of the major contributions of Tanimoto[14] to the field of languages for hierarchical systems. His proposal, while also used to derive an assembly language for the PCLIP pyramid prototype designed and built (see Section 5.2.4.2) at Washington University, originally aims at the definition of an unambiguous formalism to express algorithms for hierarchical systems.

Hierarchical cellular logic addresses hierarchical systems that fall in the broad family of cellular systems. In the naming convention of this book, this perfectly translates to homogeneous compact systems (see the taxonomy of Chapter 4). Within this class, HCL offers a tool to formulate precisely and concisely algorithms in a rather machine-independent way.

The HCL pyramid algebra will be described in the following five subsections; the first gives some preliminary definitions, the second introduces the primitive operators, called cellular logic operators (CLOs), the third expands on the latter with some advanced operators that can be consistently used at a somewhat higher level of expressiveness. The fourth subsection highlights the relationship between HCL and mathematical morphology. Although defined to be an algebra, HCL allows one to express rather complex algorithms. A hint to the implementation of a few representative ones is given in the fifth subsection.

9.1.3.1. Preliminary Definitions

The basic notion underlining the HCL construct is that of *hierarchical domain*. A hierarchical domain is a set of *cells*, where each cell is defined by a triple $(k, i, j,)$; the obvious interpretation of such a triple is that cell (k, i, j) is located on level k, at row i and column j of the mesh which makes up that level. This definition implicitly assumes a mesh topology for the set of cells of

a level; while this is the most common assumption in homogeneous hierarchies, alternative definitions are possible with different tessellations of the plane.[15] A hierarchical domain of $L+1$ levels contains cells (k,i,j) such that $0 \leq k \leq L$, $0 \leq i < 2^k$, $0 \leq j < 2^k$. Level 0 is the vertex of the domain, while level L is the bottom layer of cells.

The definition of hierarchical domain is completed by the notion of hierarchical neighborhood of a cell—that is, of those cells adjacent to the "home" one in the same level according to 8-connectivity and adjacent in the level above and below according to the quad-tree topology. Therefore, the hierarchical neighborhood of a cell consists of 13 more cells as depicted in Figure 5.4.

In HCL, the basic "object" available is the *pyramid*. A pyramid is defined as a *function* that maps each cell of a hierarchical domain to a value. Two cases are special, namely *bit pyramids* and *byte pyramids:* in the former, the codomain is [0,1], in the latter [0–255]. Among the set of possible pyramids, *constant pyramids* are those functions that produce the same value everywhere in the hierarchical domain (e.g., 1 stands for the constant pyramid of value 1); a peculiar case is that of *plane* pyramids, which are binary pyramids with a value of 1 on a single level k and zero elsewhere, and are denoted P_k.

An ancillary definition to hierarchical neighborhood is that of *pattern*. A pattern is a function that maps the hierarchical neighborhood of a cell into a set of values, each drawn from the set [0, 1, D]. The value D is a notational convention for a "don't care" value, and some of the elementary operators to be described work on the extended domain [0, 1, D].

9.1.3.2. Elementary Operators

Pyramids are not data objects within HCL, although they do produce values in hierarchical domains; so, rather than operators on pyramids, one should properly speak of functional composition. To make the discussion less verbose, the term *operator* will be used in the following in the usual interpretation; while formally not correct, this assumption should not diminish the coherence of the presentation.

Elementary operators are the core of the pyramid algebra. A minimal set includes two special-purpose cellular logic unary operators AND_MATCH and OR_MATCH, the standard Boolean NOT, and binary operators AND, OR, and restriction. As customary, the derived difference $-$, XOR, and NAND operators are also available. The two cellular logic operators have the following syntax:

AND__MATCH(Pattern, Bit__Pyramid) (9.13)
OR__MATCH(Pattern, Bit__Pyramid) (9.14)

While defined on two operands, such operators actually work on a single bit pyramid and can thus be considered unary transformations. Their interpretation has already been anticipated in Chapter 5, in the description of the CLOs of the PCLIP pyramid prototype. It is repeated here briefly for convenience. The output of the AND__MATCH operator is a bit pyramid: the value at a cell is 1 iff all the couples of values of cells and associated pattern digits in the hierarchical neighborhood are matched or the pattern digit is *D;* the output value is 0 if at least one couple has a mismatch. The bit pyramid which is the output of the OR__MATCH operator has value 1 at those cells where at least one couple of values match, provided that the pattern digit is different from the ''don't care'' condition *D;* the output cell has a value of 0 if no such match occurs.

The other unary and binary operators have the usual interpretation. The ''restriction'' operator instead deserves a few comments. It is a binary operator that applies a functional transformation $F(\ )$ to a generic pyramid X (binary or byte), according to a controlling binary pyramid Q, denoted with the following syntax:[14]

$$[F\backslash Q](X) = (F(X) \cap Q) \cup (X \cap -Q) \tag{9.15}$$

That is, the functional transformation $F(X)$ is carried out only where the binary pyramid Q gives a value of 1, while the previous value of X is retained elsewhere. This notation captures the semantics of the enabling–disabling mechanism typical of SIMD homogeneous architectures. It is the equivalent of the **where...do...end** construct for conditional execution of the formal language $\mathcal{L}$ introduced in Section 9.1.2.

The generic functional transformation $F(\ )$ can be a standard Boolean operator as well; so it is possible to write the XOR operator also as a restriction of a unary NOT operation:

$$[-\backslash Q]\, X = (-X \cap Q) \cup (X \cap -Q) \tag{9.16}$$

While in this case $[-\backslash Q]\, X$ is equivalent to $[-\backslash X]\, Q$, the restriction operator is not commutative in the case of a general transformation $F(\ )$.

9.1.3.3. Advanced Operators

The operators defined in the previous section can be used to obtain more powerful pyramid transformations by functional composition. It is useful to introduce a set of shorthand conventions to denote such functional compositions and call them *advanced operators*.

The first advanced operator considered is the *iteration* of a function F by a finite number n of compositions:

$$F^n(X) = F(F^{n-1}(X)) \qquad \text{where } F^0(X) = X \tag{9.17}$$

One can also define the *transitive closure* $F^*(\ )$ of the functional transformation $F(\ )$ of pyramid X as

$$F^*(X) = \begin{cases} F^m(X) & \text{if } \exists\ m : F^m(X) = F^{m+1}(X) \\ \text{undefined} & \text{otherwise} \end{cases} \tag{9.18}$$

The transitive closure operator is a generalization of the recursive near-neighbor operation introduced in Chapter 5: indeed, the transformation $F(\ )$, although based on local computation, is not limited to Boolean near-neighbor operations but can be any valid HCL transformation, with X being any pyramid, not just a bit pyramid. The iteration of $F(\ )$ is carried out until a stable state is reached; situations exist in which $F^*(\ )$ is not defined, because the transformed pyramid switches between configurations at successive applications of $F(\ )$.

An example of transitive closure that does terminate in a finite number of steps is

$$[\text{OR\_\_MATCH}(\textit{Pattern\_\_Children})\backslash(1 - Q_L)]^*(X) \tag{9.19}$$

This expression takes as input a bit-pyramid function X, applies to it the neighborhood operator OR__MATCH with the pattern *Pattern__Children* specifying quad-tree cells in the lower layer of the hierarchical neighborhood of the home cell, restricts the transformations to the values above the base $(1 - Q_L)$, then iterates until no changes occur, thus building an "OR" pyramid out of the values in the base of pyramid X. Because the transformation extends upward the information in the base, it is bound to terminate after at most L iterations and could therefore also be written as

$$[\text{OR\_\_MATCH}(\textit{Pattern\_\_Children})\backslash(1-Q_L)]^L(X) \tag{9.20}$$

This example introduces other advanced constructs. They are used for building pyramids from binary images. Given a bit pyramid B defined to be zero in the hierarchical domain except for level L, the ''AND'' pyramid and the ''OR'' pyramid are defined by

$$\text{AND\_\_PYR}(B) = [\text{AND\_\_MATCH}(\textit{Pattern\_\_Children})\backslash(1-Q_L)]^L(B) \tag{9.21}$$

$$\text{OR\_\_PYR}(B) = [\text{OR\_\_MATCH}(\textit{Pattern\_\_Children})\backslash(1-Q_L)]^L(B) \tag{9.22}$$

The first expression generates a pyramid by mapping disjoint connected components in the base level into disjoint connected components in upper levels until a level is reached where they are eliminated. The apex takes the value 1 iff $B = Q_L$. The second construct instead tends to merge isolated components and always produces a bit pyramid up to the vertex of the hierarchical domain. Another way to see these operators is to consider them an instance of the larger class of COUNT__OF pyramids: a COUNT__OF pyramid is a bit pyramid in which a cell is set to 1 if its extended neighborhood in the lower layer has a specified number of cells set to 1. If the extended neighborhood is limited to the quad-tree topology, as is the case with pyramids in HCL, at most four COUNT__OF pyramids are possible: the COUNT__OF__1 pyramid is the OR pyramid and the COUNT__OF__4 is the AND pyramid. The other two COUNT__OF pyramids can be obtained with slightly more complex HCL constructs.

The operators just described can in turn be used to define other advanced operators, which are based on a mixed flow of processing, as opposed to the bottom one of the COUNT__OF pyramid. The basic operator for top-down processing is named Project( ) and can be defined by

$$\text{Project}(X) = \text{AND\_\_MATCH}(\textit{Pattern\_\_Parent}, X) \tag{9.23}$$

where *Pattern__Parent* is 1 at the ''parent'' cell and D at all other cells. The expression has the effect of copying the value of the ''parent'' cell at the ''home'' one. Project( ) can be combined with the bottom-up pyramid building operators to define two more complex operators, where the flow of data involves both upward and downward data movements: they are the *pyramidal overture* of order k and the *pyramidal fermeture* of order k.

The pyramidal overture of order k is defined as

$$\text{Overture}_k(X) = \text{Project}^k\ (\text{AND\_\_PYR}(X)) \tag{9.24}$$

With k set to 1, the effect of pyramidal overture is to clean the binary image resulting from AND__PYR (0) of those 1-cells that have O-siblings. Further iterations remove more 1-cells.

The pyramidal fermeture of order k is defined as

$$\text{Fermeture}_k(X) = \text{Project}^k\ (\text{OR\_\_PYR}(X)) \tag{9.25}$$

and its effect is complementary to that of $\text{Overture}_k(X)$, since in this case any 1-cell causes its siblings to become 1-cells as well.

9.1.3.4. HCL and Mathematical Morphology

As already anticipated in Chapter 5, in the discussion of the CLOs which are the embodiment in the PCLIP system of HCL, a close relationship exists between mathematical morphology[16] and the pyramidal logic just described. This relationship is discussed here to a certain extent.

The basic transformation of mathematical morphology is Minkowsky's ''hit-and-miss'' transform. It is defined as follows: let X be a region to be analyzed, $\bar{X}$ its background, B the structuring element partitioned into two subelements B^1 and B^2 so that $B^1 \cup B^2 = B$ and $B^1 \cap B^2 = \emptyset$. Moreover, it is possible to define the translation of B that maps the reference point to a given point z and denote it with B_z. The hit-and-miss transform of the region X by the structuring element B is the set of point z for which the following holds:

$$X + B = \left\{ z : B_z^1 \subset X,\ B_z^2 \subset \bar{X} \right\} \tag{9.26}$$

The interpretation of this transform is straightforward. The well-known ''erosion'' and ''dilation'' operators can be derived from the hit-and-miss transform. The ''erosion'' of X with a structuring element B is the special case of hit-and-miss in which $B^1 = B$ and $B^2 = \emptyset$.

The matching operators of HCL can be referred to mathematical morphology in the following way. The support of the transformations, instead of the continuum bidimensional plane, is a digitized three-dimensional space. The structuring element B is confined to the set of cells previously defined as a hierarchical neighborhood, with the origin chosen in the ''home'' cell. These cells are now partitioned into three subsets, namely B^1, B^2, and B^D: B^1 is the

set of 1-cells, B^2 is the set of 0-cells, and B^D is the set of cells where the "don't care" condition D is specified. With this choice, the AND__MATCH (*Pattern, X*) operator can be restated as the hit-and-miss transform of the bit pyramid X by the structuring element *Pattern:*

$$\text{AND\_\_Match}(Pattern,\ X) = X + (B^1 \cup B^2) \tag{9.27}$$

The set B^D is simply excluded from the structuring element used in the execution of the hit-and-miss transform. Due to the completeness of the hit-and-miss operator in mathematical morphology, the AND__Match can be used in HCL to implement all the canonical operators.

In accordance with the stated equivalences, it can be said that HCL is a special generalization of the basic operators of mathematical morphology, with extensions to hierarchically structured digitized spaces.

9.1.3.5. Examples

The application of the elementary and advanced operators of HCL allows us to efficiently code even algorithms of moderate complexity. Tanimoto[14] gives a number of examples, among which are quad-tree construction, bit counting, binary edge detection, bright spot detection, thresholding, and median filtering. One such case is adapted here to highlight the transitive closure of a functional transformation and to stress the importance of the hierarchical domain.

Let us the consider the problem of labeling the connected components of a binary image I. We assume that we have available one byte pyramid P, which is zero except for the base level, where each cell has a distinct value (e.g., the concatenation of its i,j coordinates; such a value can be easily obtained through HCL constructs). Also, we assume that we have defined a byte-valued function MAX (*Pattern, B*) that takes as input a byte pyramid B and a pattern and that produces as output a byte pyramid which has at each cell the maximum value of cells in B belonging to its neighborhood as restricted according to *Pattern* (the MAX) function can be easily expressed through HCL constructs, but is omitted for brevity). Then the following expression labels the connected regions of the input binary image I:

$$[\text{MAX } (Pattern\_\_Brothers)\ (Q_L nI)^*(P) \tag{9.28}$$

where *Pattern__Brothers* specifies the 8-connected cells belonging to the same level as the home one. It is easily verified that each connected component in I

receives as label the largest value of P contained within the region itself. The actual number of iterations depends on how compact each region is and is roughly proportional to the diameter of the region. Indeed, if the connected components are quite compact (no holes or elongations), it is worthwhile exploiting a hierarchical mode of propagation that speeds up the labeling process. To do so, one first builds the AND pyramid of the input image I and then lets the MAX( ) function propagate its values within the hierarchical connected domains having their support in I in the base level. The resulting HCL expression is

$$[\text{MAX}(Pattern\_\_ALL)\backslash\text{AND}\_\_\text{PYR}(I)]^*(P) \tag{9.29}$$

where *Pattern__ALL* identifies the complete hierarchical neighborhood of the "home" cell. This is an instance of the principle of using the pyramid topology to speed up data transfers through hierarchical links.

9.1.4. HCL and the C Language

Tanimoto's pyramid algebra introduced in Section 9.1.3 allows us to describe pyramid algorithms concisely and in a rather machine-independent way. However, it cannot be considered a language in the usual sense because it lacks many of the features common in programming languages, such as explicit type definition, variable declarations, and the control structures for structuring programs.

An attempt to embed its basic constructs in a standard serial language has been pursued by Pfeiffer,[16] who chose the C language as the host environment. The motivations for this effort are to provide both the concise algorithms description possible through HCL and the more readable format of a program coded in a well-known and well-understood standard language.

The approach chosen by Pfeiffer for merging HCL into C resembles the collection-oriented programming style described in Section 9.1. The only parallel data type is the pyramid, and the programmer has no other tool to organize data in the parallel environment. This limitation is due to the close relationship among HCL, the pyramid machine prototype PCLIP, and the new extension to C. The extended HCL is intended primarily for programming that machine, although the language could be used on other architectures as well (this is only feasible if the target machine efficiently embeds pyramid operations; see Chapter 7). HCL operators are adapted to act on pyramids to suit the C language syntax. The C control structures are not given an explicit semantics in the

parallel domain, with the single exception of the **?:** operator, used to translate HCL "restriction."

9.1.4.1. Data Types

In a program written in HCL C, variables declared with the usual C data types belong to the serial environment and cannot be used to describe data allocated on pyramids. To do so, Pfeiffer introduces two special-purpose new data types, the `pyramid` and the `pattern`. Both are "first-class, elementary data types much like integers and floating-point".[16] Also, a new type specifier `bit` is added to the standard ones, as long as a length specifier; they emphasize the bit-serial processing that is used for multibit variables in the parallel environment.

Three examples of pyramid declarations follow:

```
pyramid gray_scale:8;
.
.
.
pyramid struct {
                int pixel;
                float height;} pyr;
.
.
.
bit pyramid mask;
```

The first declares a parallel variable `gray_scale` of type `pyramid` having 8 bits; the second declares a parallel variable `pyr` as a structure (composed of two standard variables) of storage class `pyramid`. Each element of `pyr` on all planes contains an element of `pixel` and an element of `height`. The third example shows the use of the type specifier `bit` to create a Boolean parallel variable of pyramid type.

The declaration of a variable in the storage class pyramid signals that it is a parallel variable and that it must be allocated on the whole target machine, that is, on all PEs of all planes. Thus, if HCL C is used on a 128×128 base pyramid machine, pyramid variables consist of a stack of eight meshes.

A few predefined constants also have the type specifier `pyramid`. They are `true`, `false`, `Level`, `x`, and `y`. The first two are Boolean pyramids, the third asso-

ciates to every element in a plane the number of the level, and the latter two give the position of each element in the level it belongs to.

The second major addition to the C language is the pattern data type. Variables declared as patterns in HCL C have the same structure of patterns in HCL—that is, they consist of 14 elements—ordered according to the scheme of Figure 5.4. Being a type specifier, the key word pattern can be used as in the following program fragment:

```
pattern NorthWest, West;
.
.
.
NorthWest = DDDD1DDDD DDDD D;
West = DDDDD1DDD DDDD D;
```

The domain on which pattern variables are built is the set $\{0, 1, D\}$, where D indicates the "don't care" condition. A specific ordering is associated with the element of this set: $0 < 1 < D$. The purpose of this ordering is to define some operations on patterns, as will be described next.

9.1.4.2. Operators

In extending the semantics of the standard C operations to the new data types, the designer of HCL C has applied the principle of "orthogonality." This consists of the overloading of existing operations to accept as operands the new data types without introducing *ad hoc* operators.

a. Expressions As an example, arithmetic operations involving pyramids are automatically interpreted to act on each component of the parallel variables on all layers, according to a strict SIMD processing modality and to the collection-oriented programming style described in Section 9.1. Length conversion is automatically handled with proper rules. The following program fragment helps explain the handling of such cases:

```
pyramid gray_scale:8, input_pixel:6;
bit pyramid mask;
.
.
.
gray_scale = input_pixel + mask;
```

The parallel Boolean variable mask is extended to a length of 6 bits to match the length of the first operand in the addition. The extension consists of 5 leading bits set to 0 (sign extension is used with all other types of pyramid variables). The sum is calculated and the 7-bit result is sign-extended to 8 bits, to be stored in the left-hand variable gray-scale.

With reference to the other operations, analogous rules apply. In particular, comparisons between pyramid variables produce bit pyramid results.

The overloading of standard operators for patterns has been investigated by Pfeiffer.[17] Having introduced an ordering in the elements of the domain {0, 1, *D*} of patterns, Pfeiffer proposes to define the C language bitwise logical OR operator | between two patterns: the logical AND **&**; and the unary negation **!** with the usual rules for 0 and 1, and with *D* being the negative of itself. The definitions of the algebra of patterns allow coding arrangements of near-neighbor combinations.

The two basic operators of HCL, AND__MATCH and OR__Match, are supported in HCL C as unary operators acting on bit pyramid variables. Their syntax is highlighted in the following HCL C statements:

```
bit pyramid mask1, mask2, mask3;
pattern West;
.
.
.
mask1 = &⟨West⟩ mask3;
.
.
.
mask2 = mask1 & |⟨West⟩ mask3;
```

The two executable statements would be written, using HCL notation, as

```
mask1 = [AND__Match(West)] (mask3)
.
.
.
mask2 = mask1 AND [OR__Match(West)] (mask3)
```

Pfeiffer defines two patterns as ''well formed'' if no two elements are respectively 0 and 1. The admissible couples are therefore 0*D*, 1*D*, 11, and 00. With this definition and according to the pattern algebra, and **&** (and |)

operator of patterns are distributive with respect to the **&** (and |) operator of bit pyramids. In the program fragment

```
bit pyramid pyr1, pyr2;
pattern locat1, locat2, locat3;
.
.
.
locat1 = 11111111 DDDD D;
locat2 = DDD111DDD DDDD D;
pyr2 = (&⟨locat1⟩ pyr1) & (&⟨locat2⟩ pyr1);
.
.
.
locat3 = locat1 & locat2;
pyr3 = &⟨locat3⟩ pyr1;
```

the two bit pyramid variables pyr2 and pyr3 are identical.

Data movements are realized through elementary near-neighbor operations. From HCL we can easily verify that shifting of a bit pyramid variable by one position in any direction of the extended near-neighbor can be accomplished with a single AND__Match operation. As an instance, the following statement executes such a shift in direction north.

```
& ⟨DDD1DDDDD DDDD D⟩ pyr
```

A shorthand notation for this expression is N(pyr). There are 12 other such functions in HCL C.

b. Control Structures As regards control structures, the C language **?:** operator is the only one for which there exists an explicit interpretation in the parallel domain. With the terminology introduced in Section 9.1.2 while describing the abstract language $\mathcal{L}$, this operator relates to the definition of the conditional execution **where...do...end** construct of Eq. (9.10). The expression

```
cond ? exp1 : exp2
```

is interpreted with the usual meaning if cond is a serial variable, while it is interpreted in the parallel domain if it is a bit pyramid. In this case, exp1 is executed by processors where cond is true, exp2 where cond is false. We note

that such a statement is slightly more general than the **where...do...end** construct of $\mathcal{L}$, since it specifies the action to be taken with processors that do not qualify for cond. Yet, it is less than a parallel **if-else** selection, because exp1 and exp2 are constrained to be expressions and cannot be structured blocks. Also with reference to the restriction operator of HCL [see Eq. (9.15)], the modified **?:** operator is more flexible, as just explained.

The iterative constructs of the C language **for, while,** and **do-while** retain their semantics in the serial domain. Pfeiffer avoids extending explicitly their semantics to the parallel environment (as was proposed in the $\mathcal{L}$ languages). Instead, he introduces a special function **nrz(),** which accepts a pyramid variable as argument and returns a string of bits (a byte in an eight-level pyramid). Each bit is associated with one level of the pyramid variable, and it signals if the variable is set to zero on all elements of the corresponding level. The string of bits can be further manipulated with standard C language bitwise operators to produce a Boolean value; this can be the argument of a test in a serial construct, such as the terminating condition of **for, while,** and **do-while.** The following program fragment simulates the transitive closure expression of Eq. (9.19) that builds an OR pyramid from a Boolean image located on the bottom layer of a pyramid variable.

```
main()
{
    bit pyramid OrPyr, OldPyr, Image;
    pattern Children = DDDDDDDDD 1111 D;
    :
    OrPyr = (Level = 0) ? Image:false;
    OldPyr = false;
    while (|| nrz(OrPyr ^ OldPyr))
      {OldPyr = OrPyr;
      OrPyr = (Level ≠ 0) ? | ⟨Children⟩ OldPyr:OldPyr;
      };
}
```

The first executable statement loads the binary image in level 0 (the base) of the pyramid variable OrPyr; the other levels of OrPyr are reset to 0 (this is achieved with the parallel **?:** construct and the constant pyramid Level). The second variable OldPyr is reset. The controlling condition of the while loop is an expression which first calculates the XOR (symbol ^) between the two parallel pyramid variables; then it reduces the outcome to a single word through the nrz() function; finally it applies the unary bitwise OR operator (symbol ||) to

verify if at least one element in the pyramid changed. If so, the body of the loop stores in OldPyr the temporary pyramid and then calculates through an OR_Match the updated version (the updating is restricted to the levels above the base). When the controlling condition is false, the OR pyramid of the input binary image is available in `OrPyr` (and in `OldPyr`).

By comparing Eq. (9.19) with this small program, it is possible to appreciate the concise notation of HCL with respect to more common languages.

9.1.4.3. New Statements and Features

The extension to the C language to support HCL so far introduced adheres quite well to the orthogonality principle. However, a number of new statements, functions, and procedures have been proposed by Pfeiffer to make the pyramid programming environment friendlier.

Beside the **nrz( )** function, two more reduction operators are proposed: **count( )** and **sum( ).** Both accept as their single argument a pyramid variable and return respectively the number of elements different from zero in the whole pyramid and the aggregate sum of all elements in the pyramid. By producing a scalar variable from a parallel one, they serve the purpose of linking the parallel environment with the controller of the system, much like **nrz ( ).**

The handling of border problems with the bit pyramid matching operations is solved by the new statement **boundary.** It allows us to specify the value of the neighborhood of pyramid nodes on the edges of the pyramid (the children in the base, the lateral border nodes, and the parent at the apex). By default, the value is constant and false(zero). It can be set to true(one) or even to the value of the closest node (with the syntax **boundary replicate**).

HCL C has also a few commands for image loading (unloading) into (from) pyramid variables and for pyramid display on output devices. The **seed** (pyr) statement allows one to create a bit pyramid `pyr` which has a single element set to 1 and the others set to 0 (the element is chosen interactively). This feature is useful for generating dynamically the starting pyramid for a coloring operation.

Many other special function primitives can be found in Ref. 15; nevertheless we do not cover them here for brevity.

9.1.5. PYR-E

The language designed for the SPHINX system by Mehat[5] belongs to the family of languages based on the C language. While a parallel implementation of Pascal[6] has been successfully used as a programming tool for the MPP

machine, the designers of PYR-E indicate two characteristics of the C language that make it more eligible for the targeted machine: a more advanced use of pointers and an augmented expressiveness in bit-oriented operations.

The latter reason is further motivated by the general approach which drives the design of PYR-E: in the bipartition of parallel programming languages introduced at the beginning of this chapter, PYR-E stays within the class of PE-oriented languages rather than collection-oriented ones. A direct expression of the behavior of the PE, including its interaction with other PEs in the architecture, is a key feature of the language. Parallelism is embedded in the language by attaching an *explicit* specification for each variable as belonging to one of the two *domains,* serial or parallel. Functions are *implicitly* overloaded by conversion rules to act on the appropriate domain.

9.1.5.1. Enabled Processor Set and the Programming Modes

In the discussion of the language, a few definitions are assumed, which are now made explicit. The term *enabled processors set* (EPS) is used to refer to that subset of all processors in the machine that are currently enabled through a proper setting of the condition register and that therefore participate in the execution of the next broadcast instruction. This is precisely the definition of context as introduced in the discussion of the $\mathcal{L}$ language (see Section 9.1.2).

Furthermore, the designers of the language introduce two modes of programming that are available in the SIMD paradigm for which PYR-E is suitable, namely a *local hierarchical* programming mode and a *global non-hierarchical* one. The distinction is relevant to the description of how the flow of control is managed in the parallel environment.

The local hierarchical mode ensues through the use of the condition register of each PE and the corresponding definition of the current EPS: as a result of an operation based on *local* data, each PE establishes if it belongs to the next EPS, and the PEs of the current EPS are split into two disjoint sets. The controller of the machine uses this partition to broadcast (serially) two different instruction streams, and the execution of each such stream can give rise to further subdivisions of the various EPSs, thus realizing a *hierarchical* flow of executions: branches in the flow of the program are the outcome of a local test at each PE.

On the contrary, the global non-hierarchical mode controls the flow of execution using the value of the global OR, which collects its value from the whole current EPS; it is the contents of the status registers of the EPS, whichever way it is obtained, that determines the branch in the program flow in the

controller, and the resulting stream of instructions is carried out by the EPS, without the creation of any hierarchical subdivision among PEs.

The control structures of the standard C language are reinterpreted in PYR-E to operate on the PEs according to one of the two modes just described.

Another peculiarity of PYR-E stems from the pyramid architecture of the SPHINX machine for which it has been conceived. Since the language aims at making explicit the interaction among PEs, it requires proper constructs for data exchanges both within a layer of the pyramid and between adjacent layers. While the former type of communication is rather standard, the second one has to cope with the possible Multi-SIMD operative modality of the underling machine, and makes explicit the synchronization mechanism adopted. In the terminology of Mehat, *transmission of control* is the strategy on top of which the language builds its constructs to express interlayer data exchanges. This latter topic will be described in Section 9.2.

9.1.5.2. Data Types

The concept of *domain* is used to establish whether the declaration of a variable affects the controller or the parallel structure. In the former case, the domain is *serial,* and variables of this kind only exist within the controller; in the latter, the domain is *parallel,* and an instance of each variable of this kind exists in every PE. The domain is specified with the key words *serial or parallel* in the declaration of each variable:

```
serial type-specifier variable
parallel type-specifier variable
```

The domain is further extended to pointer variables as well. Three cases are relevant:

```
serial type-specifier serial * pointer-variable
parallel type-specifier serial * pointer-variable
parallel type-specifier parallel * pointer-variable
```

The first involves only controller variables and deserves no discussion. The second case declares a controller variable used as a pointer in the memory of all PEs; this corresponds to a strict SIMD global addressing modality, since it is the controller, through the variable declared as a pointer, that establishes the memory address to be used by all PEs. The third case only involves vari-

ables in the parallel environment: a local variable in the PEs is used to locally address the memory, thus realizing a form of autonomous memory addressing which is only possible because of the specialized hardware of the SPHINX PE.

The types available in the parallel domain are all the basic and derived types of the standard C language, with the associated storage class and type specifiers. The only deviation is the possibility of adding a *length specification,* which sets the number of bits of the associated variable. This feature, available in the C language only for members of structures and unions (the *bit-fields*), makes explicit the bit-serial nature of the processing modality supported by PYR-E. The length is specified also for constants and *pseudoconstants;* these are parameters which change with the level of the pyramid, but are otherwise constant in the associated PEs, as in the following case:

```
int pixel: 8+FLOOR
```

Syntactically, this declaration involves an expression, and the parameter FLOOR is instantiated only at run time by the controller.

9.1.5.3. Operators

The whole set of operators, control structures, and functions or procedures of the standard C language is available in PYR-E. Their semantics is conditioned by the domain of the associated variables; in the *serial* environment, nothing differs from the usual interpretation, while in the *parallel* one a new meaning is attached to each construct, as we briefly show.

a. Effect of the Domain on the Semantics of Operations. The construction of expressions is the first example. Whenever a variable, declared as *parallel,* is used in forming an expression, the whole expression is *implicitly* converted to the parallel domain. *Serial* variables are transformed into *parallel* ones by proper actions from the controller, which broadcasts to all PEs the pertinent value. Assignments are handled accordingly: a *serial* variable can only be the recipient of a *serial* expression, while a *parallel* variable forces the right-hand side of the assignment to be converted to the parallel domain so that each PE can store the appropriate result. As a consequence, the assignment of a *parallel* variable or expression to a *serial* variable is illegal; the domain conversion from parallel to serial is possible only in a very special case, covered later.

A special treatment, deriving from the parallel context in which they are declared, is that of variables for which a *length specification* has been given. The usual conversion routines of the standard C language would make poor use

of memory space. The general rule adopted instead is to keep the size of the result of an expression to the minimum necessary to correctly represent it.

b. EPS and the Local Hierarchical Programming Mode Among the control structures of C the Boolean test **if-then-else** gives rise to the local hierarchical programming mode, where the flow of control is partitioned according to the outcome of an expression evaluated in each PE. The effects of the parallel domain on the **if-then-else** control structure are straightforward. The syntax is the usual one:

```
if (test_expression)
    {block_true}
else
    {block_false}
```

and the structure receives the new interpretation if and only if test_expression belongs to the parallel domain. In this case, the semantics is the following: enabled PEs, where the expression of the test evaluates to true execute the first block of instructions; the others execute the second one. In other words, all PEs in the current EPS evaluate test_expression, which produces as a result a single-bit tmp, and save in memory the content c of the masking register M; then M is loaded with tmp, thus changing EPS, and the PEs belonging to the new EPS execute block_true; M is now loaded unconditionally with the logical AND between c and not(tmp), which resets EPS to the complement of the set just active; PEs in this EPS execute block_false; lastly, M is restored unconditionally to its initial value c.

The only difference between the **where...do...end** construct of $\mathcal{L}$, described in Section 9.1.2, and the **if-else** statement of PYR-E is in the **else** clause of the latter, which is missing in $\mathcal{L}$. The semantics, though generated differently, is the same.

The multi selection operator **switch-case** has semantics quite different in the parallel domain with respect to the serial case. It retains the usual syntax:

```
switch(val)
  {
  case val_1 : {block_1}
  .
  .
  default    : {block_d}
  }
```

where val is an integer expression belonging to the parallel domain and val_1,..., val_n are integer constants. The interpretation of the construct is as follows: the initial EPS is saved in memory and emptied; the resulting (initially empty) EPS is augmented by the sets of processors in which $val = val_i$ and which execute the instructions of $block_i$; when the label default is reached, the new EPS is the complement of the current EPS and executes $block_d$. Finally, the initial EPS is restored from memory.

The notable difference with serial semantics is the ordering introduced by the enumeration of the labels and by the handling of default. This clause is executed where none of the preceding comparisons hold, while in the serial case it is activated when none of the comparisons hold anyway. As a consequence, normally the **break** statement, used in the serial construct to close the otherwise sequential execution of the alternatives, is missing unless explicitly required by the logic of the instructions.

No definition of the multi select construct is available in $\mathcal{L}$. A straightforward handling of the context stack according to the actions of the EPS of PYR-E is one of the possible definitions (we note that there seem to be more semantics for such a construct).

c. EPS and the Global Nonhierarchical Programming Mode. The block control structures **do, while,** and **for,** if executed in the parallel environment, rely on the second programming style, the globally controlled one. Implicitly, a conversion from the parallel to the serial domain is involved in this programming mode, since the next instruction broadcast by the controller depends on the status of the PEs. PYR-E uses three new operators to execute this domain conversion: **any**(expr), **all**(expr), and **none**(expr), where expr is an expression in the parallel domain, and the result of the operators is a value in the serial domain, that is, in the controller. The semantics of **any** is obviously that the value returned to the controller is true if expr evaluates to true in at least one PE of EPS; **all,** in turn, returns true if expr is true in all PEs of EPS; **none** is the dual case of **all,** and returns true if expr is false in all PEs of EPS.

With these operators, the **while** control structure

```
while(expr)
  {block}
```

where expr is defined in the parallel domain, is interpreted as

```
while(any(expr))
  if (expr) {block}
```

that is, the instructions included in block are excuted by the PEs of EPS for which `expr` is true as long as this set is not empty. Procedurally, PEs in EPS save M unconditionally in `c`; then they evaluate `expr` which yields a bit `t`; M is loaded with `t`, and the new EPS executes `block` if it contains at least a PE; then the flow goes back to the previous stage, otherwise the structure is exited and EPS is restored to the initial EPS by reloading unconditionally M with `c`. Analogous transformations and interpretations hold for the other two block control structures.

These operators highlight the different flavor of the PYR-E language with respect to $\mathcal{L}$. Instead of using an implicit predicate to express the global-OR function required by the controller, PYR-E uses an explicit means for switching between the two environments. This follows the basic principle that sustains PE-oriented languages, namely to make explicit the role of the PEs and that of the controller.

The C language statements **break, return, continue,** and **goto,** which modify the flow of execution by causing an exit from the control structures, have to be reinterpreted as well. The effects of **break, return,** and **continue** are explained in terms of the EPS of the control structure from which the flow exits, as follows:

```
while(cond)
   {
   if(test1)
          {if(test2)
                 break;
           else
                 {block1}
           }
   else
           {block2}
   }
```

where cond, $test_1$, and $test_2$ belong to the parallel domain. Each evaluation of cond generates a set of processors that participate in the execution of the while block; let this set by EPS_1. The evlauation of $test_1$ partitions EPS_1 into EPS_{11} and EPS_{12}; processors in EPS_{12} execute $block_2$, processors in EPS_{11} evaluate $test_2$, which further partitions EPS_{11} into EPS_{111} and EPS_{112}. Processors in EPS_{111} execute the **break** statement; the effect is that they are removed from EPS_{111}, EPS_{11}, and EPS_1 as well, thus realizing an exit from the control struc-

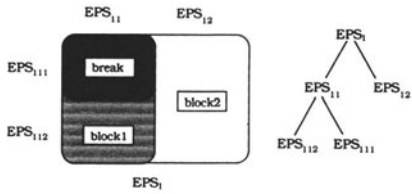


Figure 9.1. Graphical sketch of hierarchical arrangement of the EPSs generated by the two-nested **if-else** constructs. The execution of the break instruction within EPS_{111} removes this set of PEs from the whole hierarchy.

ture. In Figure 9.1 a graphical representation of the involved hierarchy of EPS is given.

Similar to this is the behavior of **return,** which obviously acts on the instruction sequence within functions. As to **continue,** its effect is the elimination of the PEs that execute it from the EPSs nested within the control structure (so these PEs skip the sequence of instructions in the rest of the block) and the generation of a new EPS, which will be fused with the remaining PEs that undergo the evaluation of the terminating condition of the block. In this example if the statement **break** were replaced by **continue,** processors in EPS_{111} would be removed from EPS_{11} but not from EPS_1 and would participate in the successive evaluation of cond.

The semantics of the break statement here described closely resembles the **escape** statement of $\mathcal{L}$. In the case of nested EPS, however, the syntax of **escape**$_i$ in $\mathcal{L}^e$ is much richer and more explicit.

The only statement for which no interpretation is proposed in the parallel domain is **goto.** The label referenced with a **goto** can be associated with a statement contained within a block either parallel or serial; while in the latter case no problem arises, in the former difficulties arise in the resulting EPS. In PYR-E the behavior of the program is not specified in such a case.

9.1.5.4. New Statements and Features

As in the other languages considered, PYR-E introduces a number of special functions; even if they technically do not belong to the formal specification of the extended C language, they are a crucial part of the new language. These special functions are used for communication among neighbors and for the management of interlevel activity, such as task creation and ''control transmission.''

PYR-E distinguishes interlevel and intralevel communications. The former involve the interaction of tasks running on different planes of the pyramid under different controllers and therefore requires a form of synchronization; the latter rely on the routing capabilities of PEs and are easily embedded in the language with the aid of the following special functions:

```
North (par_expr, disp)
East (par_expr, disp)
South (par_expr, disp)
West (par_expr, disp)
```

The use of these functions obeys two simple rules: transmission is unidirectional, and it is a read operation executed by the destination PE. The two arguments of the functions have the following meaning: par_expr is an expression in the parallel domain, which specifies the destination variable where the datum will be stored; disp is the displacement in the specified direction where the source PE is located.

Unlike par_expr, disp can belong either to the serial or to the parallel domain. No special interpretation is required if it is a value from the serial domain, that is, from a controller variable. The statements

```
serial int step;
parallel int pixel:8;
  .
  .
  .
step = 2;
pixel = East(pixel,step);
```

are the simple code to specify a westward shift by two positions for the variable pixel in the whole pertinent plane. However, if the displacement is itself a variable in the parallel domain, the intended behavior changes, as in the code

```
parallel int pixel:8, step;
  .
  .
  .
pixel = East(pixel,step);
```

In this case, each PE reads the variable pixel from a source PE located to its east at a displacement which varies with the destination PE, since the variable step is allocated in the whole plane. It can be easily verified that the execution of these statements causes the controller to enter the ''global test'' modality. Indeed, the following instructions are issued: each PE copies step into a local variable count; variable pixel is loaded, bit per bit, in the shift

register devoted to near-neighbor communication; it is then moved in the appropriate direction by one position; all PEs decrement variable count by 1; PEs where count gets to zero read in the shift register contents and store the value in memory at the address of variable pixel; then the controller issues the instruction **any**(count) and establishes whether any destination PE still has to receive their intended value; the preceding steps are iterated until **any** returns false. The use of the domain conversion statement **any** signals the global test modality, which turns this data movement into a dynamic, data-dependent operation.

Although the four communication functions imply unidirectional transmission, the parallel domain associated with the displacement does not prevent using signed variables. This actually modifies the behavior of the invoked function, and the controller has to iterate twice the procedure just outlined to cover the two possible directions of movement.

The interpretation of the data movement functions in terms of ''read from'' operations allows one to specify easily the effect of the status of PEs involved in the data transfer. Only PEs in EPS read in the transmitted value; the remaining PEs do not store the value contained in the shift register, but they do transmit the content of the variable involved in the movement. It is the responsibility of the programmer to handle the anomalies arising on the frontier of EPS as a consequence of this protocol. PEs outside EPS do not execute the instructions resulting in variable assignments, and therefore they can end up transmitting garbage data when a data movement instruction is issued from the controller. A direct consequence of this effect is the different result produced by the program segments

```
parallel int pixel:8;
serial int disp1, disp2;
  .
  .
  .
pixel = West(pixel, disp1 + disp2);
```

```
parallel int pixel:8;
serial int disp1,disp2;
  .
  .
  .
pixel = West(pixel,disp1);
pixel = West(pixel,disp2);
```

In the second program fragment, the first assignment might correspond to PEs outside EPS; the value thus transmitted by such PEs at the next instruction is not reliable, since it is obtained by a memory location which has not been properly loaded.

Programming More Planes. The description of the language thus given gives no hint of the pyramid machine structure (the designers of PYR-E explicitly stress this point as the basis for the portability of PYR-E on other SIMD machines, such as the Connection Machine). In fact, algorithms that use the pyramid structure require two more types of extension to the standard C language, namely a declaration of the allocation of variables on the planes of the pyramid and the protocol for data transmission between adjacent layers.

As to variable allocation, PYR-E distinguishes *static* and *dynamic* variables. The first is implicitly allocated in the memory of the PEs of *all* planes unless an explicit *plane delimiter* is used to limit the set of planes where the variables are created to either the base or the apex of the pyramid, as in the following example:

```
{TOP}   parallel long int sum;
{BASE}  parallel int value:16;
```

which *statically* allocates on the bottom array the parallel variable `value`, with a length of 16 bits, and the long integer variable `sum` in the topmost layer of the structure. Dynamic variables only exist within the layer where they have been allocated within the body of a function executing only on that layer, or explicitly as the result of a PEalloc memory management function.

The handling of data exchanges between planes is covered in detail in Section 9.2 in the description of the control environment hosting a program for the SPHINX bin pyramid.

9.1.6. PCL

The Pyramidal C Language (PCL) is another proposal for programming fine-grained pyramid machines with a rather high-level language based on C. It was initially presented[18, 19] as one of the tools to be developed for the software environment of the PAPIA 1 system. The programming environment of that prototype machine consists of a layer of languages, from the basic machine code used to develop the system libraries to the macroassembly language that embeds PAPIA 1 machine code into a set of mnemonics (a brief description of this environment is available in Section 5.2.4.1c. Later extensions[20] to PCL have led to an intended wider scope for the language. The following brief

description concentrates only on those parts of the language that are pertinent to pyramid machine programming.

With reference to the programming styles introduced in Section 9.1.1, PCL belongs to the collection-oriented languages. In close similarity to HCL C (see Section 9.1.4), it explicitly introduces parallel data types, among which are the notions of ''image,'' ''pyramid,'' and ''context.'' The last one is an attempt to enrich the notion of context as given by Levaire and Bougé to the hierarchical domain.

The control structures of C are partially overloaded and given a parallel meaning and partially augmented with explicit new parallel constructs.

9.1.6.1. Data Types

PCL has an explicit syntax for declaring both standard and parallel data types. We adopt the notation of Levialdi,[21] who gives a Backus–Naur definition of the declaration section of a PCL program.

⟨declaration⟩ ::= {⟨context_ident⟩} ⟨decl_specifier⟩
⟨decl_specifier⟩ ::= ⟨type_spec⟩⟨declarator⟩ |
⟨sc_specifier⟩⟨declarator⟩
⟨declarator⟩ ::= ⟨ident⟩ {⟨const⟩}
⟨type_spec⟩ ::= ⟨simple_type⟩ | ⟨parallel_type⟩
⟨simple_type⟩ ::= **character** | **integer** | **short**
integer | **floating** | **boolean** | **mask**
⟨sc_specifier⟩ ::= **auto** | **extern** | **register**
⟨parallel_type⟩ ::= **context** ⟨ident⟩; |
ima of ⟨simple_type⟩⟨ident⟩[⟨sizex⟩,⟨sizey⟩];|
pyra of ⟨simple_type⟩⟨ident⟩[⟨base⟩,⟨height⟩];|
select ⟨ident⟩[length];
⟨base⟩ ::= ⟨int_expression⟩
⟨sizex⟩ ::= ⟨int_expression⟩
⟨sizey⟩ ::= ⟨int_expression⟩
⟨height⟩ ::= ⟨int_expression⟩
⟨length⟩ ::= ⟨int_expression⟩

The basic assumption of the typing mechanism of PCL is that the parallel type is a derived type and is based on simple types. These are the standard types of the C language, with two notable additions, namely **boolean** and

mask. The former is used for parallel variables represented by a single bit; the latter has the special function to create subsets from other parallel variables and to contribute to the definition of context type.

The **context** type is a way to create a partition in a quad-pyramid data structure. Context variables consist of an ordered set of four integer values (x, y, 1, d); they represent concisely a (trunk of a) quad pyramid. The first two integers are the x,y coordinates of the apex; the third coordinate is the topmost level, and the fourth is the depth. For example, in a pyramid of eight levels (level 0 is the base), the following expressions define two disjoint contexts:

```
context Top, Trunk;
.
.
.
Top = (0,0,7,3);
Trunk = (*,*,4,5);
```

The Top context is the three-level subpyramid sharing the apex with the underlining eight-level pyramid; Trunk instead extends from level 4 downward to the base (the notation, *,* in context variables stands for all pyramid nodes of the corresponding levels). Context variables are declared within the **main** program. A default ''universal context'' is implicitly declared externally: it consists of a pyramid as large as the running environment supports.

The primary purpose of the context type is to act as the support for the two other parallel data types, **ima** and **pyra.** Parallel variables of both types can only be declared within a context. With respect to this, the context of PCL is the hierarchical counterpart of the ''shape'' concept in Thinking Machine C*. The **ima** data type declares mesh variables of the given sizes. The **pyra** type declares trunk-pyramid variables: the trunk pyramid has a linear dimension $2^{\langle base \rangle}$ in the base and $2^{\langle base \rangle - \langle height \rangle + 1}$ in the top. The following are some examples of parallel-variable declarations:

```
context Top, Trunk;
.
.
.
ima of character Pixels[256,256];
pyra of real A[8,5];
Trunk pyra of int B[8,3];
```

Variable `Pixels` is a mesh instantiated in the plane of the pyramid (the dimensions refer to the base). Variable `A` is of type **pyra** but has no context specifier and is therefore instantiated in the universal context; however, it only consists of five levels and therefore is actually a trunk pyramid. Variable `B` is defined within the `Trunk` context but is itself a subset of that trunk pyramid.

A third new parallel data type is **select.** It is used to create variables consisting of a string of bits, each bit representing a single position in the cellular logic neighborhood of a pyramid node.

According to the proposal of the PCL designer, the context concept serves a second purpose, beyond typing variables. It is a way to specify task partitions in a pyramid system. A task, before running in the system, must be associated with a context, which specifies its data environment (variables, etc.). Depending on the management of concurrence (SIMD, multi-SIMD, or MIMD), contexts can be manipulated through the associated context variables to schedule processes (''context algebra'').

9.1.6.2. Operators

PCL adopts different strategies in extending the semantics of C constructs to the hierarchical parallel domain.

Implicit overloading is used in the assignment statement. A special form of assignment is the ''selection statement'' with the following syntax:

⟨ident⟩ (⟨mask_expression⟩) = ⟨par_expression⟩

The assignment to the left-hand-side parallel variable ⟨ident⟩ is conditional on the value of ⟨mask_expression⟩. This is built with variables of type **mask** and restricts the assignment to elements of ⟨ident⟩ where ⟨mask_expression⟩ is true. We note that the ''selection statement'' modifies the set of active processors only for the execution of the assignment. With regard to the formal language $\mathcal{L}$ introduced in Section 9.1.2, it can be defined as a conditional execution of a single instruction rather than of a block of instructions.

PCL introduces three explicit parallel control structures. The first is a **where-otherwise** structure that easily generalizes the **where...do...end** of $\mathcal{L}$ and matches the **if-else** of PYR-E (see Section 9.1.5.3). The other two are **pwhile** and **pdo-puntil.** They extend in the parallel domain the corresponding standard C language construct. Both require that the controlling condition be a parallel expression of type **boolean.**

9.1.6.3. New Statements and Features

A number of special-purpose built-in functions are available in PCL for managing image I/O, for computing image statistics on the pyramid variables, etc. Among the more unusual ones are the special instructions required to activate and deactivate contexts: **open context** ⟨ident⟩ and **close context** ⟨ident⟩. They delimit the section of **main** in which the parallel variables are allocated and in which the associated task is scheduled.

Data exchanges within neighboring nodes are implemented with a set of "broadcasting" functions. A multistep **shift** (it accepts a parallel variable of type **ima** or **pyra,** a scalar displacement, and a direction) moves data horizontally; **sendup**, **sendown**, and **transf** realize vertical communications with either **pyra** or **ima** variables.

Furthermore, explicit functions implement near-neighbor operations to support pyramid machine instructions such as those of PAPIA 1 (see Section 5.2.4.1). Such functions take as operands a Boolean parallel variable and a select variable; the second one specifies which data contribute to the computation among those of the near-neighboring pyramid nodes. PCL also has an explicit support for mathematical morphology.

9.2. CONTROL ENVIRONMENT

Profitable use of a hierarchical system such as those described in this book does not depend only on the programming language available. In analyzing the various architectures, we have distinguished the proposals according to the granularity and homogeneity of the processing units. A great variety exists among them, and such differences appear not only in the hardware at processor level and their interconnection but also in the control of such different systems.

It is quite evident that control cannot adhere to a single model across all architectures. The SIMD compact pyramid requires a relatively simple control with respect to a cluster of heterogeneous small pyramids. Yet, a compact pyramid can be substantially more difficult to handle if we want to run different tasks according to the resolution, thus introducing a multi-SIMD modality. All this has consequences on the structure of the controller(s) and on the programming environment. In this section, we characterize the control environment with special regard to the multi-SIMD modality. We review the most important proposals that have been put forward to design the controllers, and we analyze the influence of the various controlling strategies on the programming environment.

9.2.1. Control of Multi-SIMD Hierarchical Systems

Hierarchical multiprocessor systems can be built to embed different forms of parallalelism. Fine-grained, massively parallel pyramids are best suited to "data parallelism," with a single task having control of all the computational resources. The multilevel structure of the pyramid allows introduction, at least in principle, a form of "task parallelism," with tasks distributed on the levels of the pyramid and interacting through the vertical communication paths. Furthermore, we can also consider heterogeneous systems, where the levels of the hierarchy are realized with processors of different granularity: usually higher levels (the symbolic layer) are managed by advanced microprocessors, and lower ones (the iconic layer) by nets of bit-serial processing elements. Sometimes, the system is also partitionable in the clusters (subpyramids), each comprising a subset of the network of bit-serial processing elements, and one master processor, in charge of higher-level routines.

We can gather such different systems under a unifying class by using the notion of multi-SIMD processing modality. It is possible to identify in the systems groups of slave processors which are only capable of carrying out, in strict SIMD mode, the instructions broadcast to them from a master processor. This master processor can be just a controller (whose only task is the dispatching to the slaves of the instruction and management of the interaction with other partners) or a working processor (in this case, it also participates with a computation task for the overall processing).

Conceptually, such a system is capable of running multiple tasks on different data and thus closely matches the definition of MIMD processing modality. We think, however, that the multi-SIMD characterization is more precise, because it retains an explicit hint of the existence of a (possibly) large set of bit-serial processing elements. This has a definite impact on the controlling strategy.

Indeed, the multi-SIMD environment can be implemented along two different dimensions in hierarchical systems: across the *resolution* and across *space*. The first case refers to the use of the pyramid as a set of meshes of linear dimension decreasing with the levels (and with the resolution, accordingly). Each plane is a SIMD mesh managed by a plane controller and acts as a single "node" in a network of linearly connected processors. In the second case, the pyramid is partitioned into clusters, each cluster covering, eventually at more resolutions, a region of the whole image (input data, in space). Such clusters are interconnected in bidimensional networks and can themselves be considered the basis for further groupings at higher levels.

The type of control required for these two forms of multi-SIMD processing tends to be somewhat different.

The linear (across the resolution) environment lends itself to the management of tightly coordinated processes that interact through massive data exchanges along the hierarchical connections. In the simplest case, each level has a single task, and therefore the overhead in synchronizing adjacent tasks on the plane controllers is low. However, synchronization of data exchanges between the SIMD layers has more stringent requirements, because it has to deal with bit-serial processing and must possibly deal with the rather sustained timing of the SIMD array. As we will see in detail in the following section, a specially critical situation ensues when the pyramid hosts mixed data movements (upward movements for data reduction functions and downward movements for distribution of results). More complex activities are possible, though less common.

The bidimension arrangements of clusters of subpyramids require a richer control strategy. The cluster has to synchronize the activities of the slave SIMD processing elements with the master processor: the amount of data is usually reduced, compared with the previous case. However, synchronization at the bit-serial level is required when two clusters have to exchange data to the iconic level (at the boundaries of the SIMD meshes). Moreover, clusters interact at the symbolic level as well, when master processors exchange aggregate partial results; communication has a much coarser granularity and less stringent timing requirements, but it demands a richer set of primitives.

From the viewpoint of control management, a very important issue is the required effort of the programmer. In coding the algorithms, the visibility of the synchronization mechanism is usually a problem, because it makes program code less readable and less portable. The second form of multi-SIMD processing mode just described is the most akin to a general MIMD paradigm. The across-resolution multi-SIMD mode allows a more compact coding of the tasks interaction, because the tasks to be coordinated are limited in number and because they are arranged topologically along a linear network of processors.

9.2.2. Multi-SIMD Control across Resolution

The following discussion is based on the experience gained by the designers of fine-grained compact-pyramid systems, such as those described in Chapter 5. We note that the SIMD modality, being the easiest from the conceptual and practical points of view, is the one actually implemented in three of the four pyramid systems that became operational, at least at the prototype level.

The SPHINX bin pyramid distinguishes itself from the outset as a system designed to be multi-SIMD. In the process of consecutive refinements of an initial specification, various versions of a controller have been investigated.[22–25] We describe them because they highlight the requirements introduced in the controlling environment by different classes of algorithms.

9.2.2.1. Unidirectional Communication

Let us consider initially a very simple algorithm, namely the computation of the histogram of all pixels in an image stored in the base of a pyramid. The algorithm is an instance of unidirectional flow of data, since the pyramid is used to perform a reduction. It counts, in logarithmic time, the number of pixels of the input image stored in the base, having a value equal to the one broadcast by a controller. The count is accumulated in the apex. In the following, we give a pseudocode (adapted from Clermont and Mérigot[22] and Méhat and Mérigot[23]) of a possible implementation on a bin pyramid with `height` levels. The base is level 0 and stores the input image `pix_value`, while the apex is level `height-1`.

```
                for i=0 to max_pix_value
                begin
{base only}       enable PEs where pix_value=i
                  count PEs enabled
                end
```

```
                {count}
{base only}
                begin
                  transmit bit to the parent
                end
{on plane L:    1≤L<height−1}
                  begin
                    for j=1 to L
                              begin
                                sum bits from left and right children
                                transmit result to parent
                              end
                    transmit carry to parent
                end
```

```
{apex only}
            begin
                for j = 1 to height
                        begin
                          sum bits from left and right children
                          store result bit in memory
                        end
                store carry in memory
            end
```

Apparently, the computation is rather uniform across the pyramid, since the role of all PEs is to gather the intermediate count computed by the children, sum it up, and transmit it to the parent. The exception is in the base, where processors do not perform an appreciable computation, and in the apex, where the computed value will be stored somewhere in the local memory. Even at such a coarse level of analysis, it is clear that the control modality cannot be SIMD. This is even more so if we break down the algorithm into its bit-serial processing and if we take account of the vertical transmission modality in the bin pyramid.

The processing element of the SPHINX (see Chapter 5 for details) executes a vertical transmission with a single read–modify–store machine cycle. The highest utilization of the pyramid machine is therefore obtained by pipelining the ascending bit-serial stream of data so as to keep the layers of the pyramid busy as soon as the information is available. If the vertical transmission obeys a producer–consumer protocol at the bit level, the occupancy diagram of the pyramid (see Figure 9.2) for the histogramming algorithm shows a remarkable diversity of instructions to be broadcast to the layers at each new machine cycle. So, an apparently simple algorithm turns out to be a demanding task on the controller.

For this reason, the controlling environment for the SPHINX machine has been designed initially as a three-level subsystem.[22] At the apex of the hierarchy is the host computer. The next level is a linear arrangement of plane controllers; each one is attached to a macro generator, which transmits the SPHINX op-codes through a FIFO queue to the low-level controllers (see Figure 5.15). These drive the meshes of PEs and are in turn interconnected linearly to perform vertical transmission synchronization. While the interaction between tasks on the plane controllers is managed through message passing, the low-level synchronization must cope with the speed of the vertical transmission channels.

Clermont and Mérigot[22] have proposed an elegant and speedy implementa-

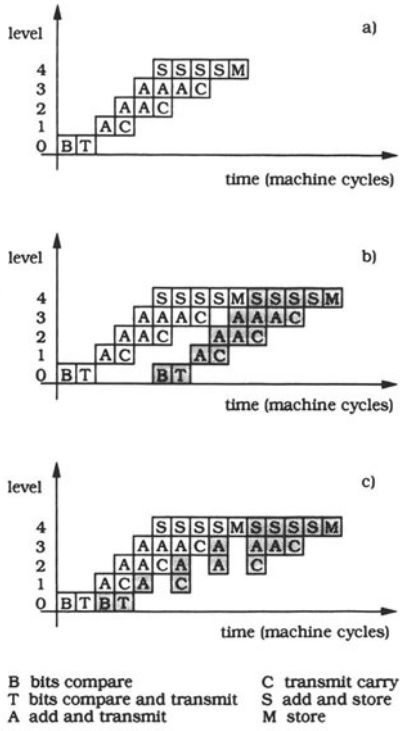


Figure 9.2. The occupancy diagrams of the bin-pyramid levels as generated by the histogramming algorithm (input pixels are 2 bits deep). (a) execution of a single comparison and upward count; (b) two such steps are pipelined for optimal occupancy of the top level only; (c) the best possible execution under the producer–consumer protocol at bit level.

tion of the low-level controllers as automata. The required hardware is minimal, and the scheme always guarantees the correct synchronization of data exchanges. Basically, each plane automaton manages the state of the buffers associated with the vertical transmission channels, analyzes the instruction to be extracted from the queue and determines locally if the instruction is compatible with the producer–consumer protocol at the current time. To reach a decision, each automaton requires some information from its two neighboring automata above and below. The state transition function that computes if the instruction can be executed is very simple and can be easily implemented.

The speed of the low-level controller is, however, limited by those pyramid operations that perform a global shift of data in either vertical direction. A simple example is the execution of a read–combine–transmit operation on all planes. According to the producer–consumer protocol, this instruction can be executed if the local input buffers are filled with data. But locally, each automaton, except the one at the apex, sees a full output buffer, and the activation

of the operations requires a top-down propagation of consensus. So, while at any time the set of automata that can schedule the instruction is a legal subset of the final one, the time required for the consensus to reach the base might exceed the read–modify–store machine cycle time, thus slowing down the machine.

This linear arrangement of automata can be used to synchronize any linear arrangements of asynchronous processing elements (asynchronous systolic arrays, etc.) by embedding properly the synchronization mechanism in the state transition function. With regard to the embedding of such a control scheme in a real system, we note that the programming environment would not be affected by the synchronization of data transmission, which is handled completely transparently.

9.2.2.2. Crossing-Data Movements and the Transmission of Control

The controlling subsystem based on the synchronizing low-level automata, though simple and efficient, cannot handle concurrent upward and downward flows of data. Méhat and Mérigot[23] illustrate the point with a modified histogramming algorithm: the count for each histogram bin has to be redistributed to the base and stored in every PE initially containing the value of the bin (associative histogram). This algorithm gives rise to two independent streams of data: the upward one, identical to the previous histogramming algorithm, and a downward one, which simply consists of copying at each level below the apex the parent's value. PEs in the base compare such a value with their pixel value and conditionally store it in memory if the two match.

Such crossing-data movements can give rise to deadlocks if the producer–consumer protocol is enforced with a single task on each layer. In the aforementioned case, a deadlock occurs between the apex layer and the one close to it as soon as the former has produced the first bit of a count and has redirected it downward. The circular link between the two tasks arises because the task on the apex is waiting for the lower one to consume the first bit of the count; the task on the lower level is waiting for the apex one to consume the data required for the second bit of the count.

The problem has been addressed by Méhat[23] introducing a new structure for the control subsystem and a new protocol to schedule data. The proposal considers the vertical channels as a nonsharable resource. Each plane has two such resources, an ascending channel and a descending one. Moreover, Méhat defines ''process'' as the set of instructions of a task that manage vertical data transmissions. According to these definitions, processes can belong to one of the following classes: ASCending, DEScending, MIXed, and NEUtral. The

first two classes cover unidirectional data exchanges, whereas the third includes processes that interact in both ways with neighboring levels of the pyramid. NEUtral processes do not execute vertical data transmissions.

The new protocol for scheduling processes in the plane controller allows in each layer an unlimited number of NEUtral processes, together with a single MIXed process or one ASCending and one DEScending process. So, each vertical resource in a plane is owned by at most one process.

The low-level synchronization is carried out by low-level controllers. The difference with the previous scheme is that such controllers are not linked to realize synchronization. Instead, they are designed to read instructions from two separate queues, one for ascending and one for descending data streams. MIXed processes use both queues. The protocol is such that a low-level controller dispatches instructions by reading from one queue as long as the interlevel resources are available. When this condition no longer holds, the controller executes a context switch and activates the single process in the level that uses the other vertical direction, by reading from the other queue.

The protocol guarantees a deadlock-free scheduling of processes, provided that the pyramid hosts at most a single MIXed process. The modified histogram algorithm would generate one such process in the apex. Deadlock can occur, however, if two or more MIXed processes are allowed to execute concurrently.

This proposal is completed with the specification of process generation. Méhat introduces the notion of ''transmission of control,'' which is a way to specify explicitly, though concisely, cooperation of processes in the programming environment. According to this proposal, processes are created where the streams of data originate and ''migrate'' through the plane controllers along with the data. As an instance, an upward data movement involves the creation of an ASCending process in the base of the pyramid as a first step. This process uses a process management primitive of the plane controller to create in the layer immediately above an ASCending process which is a replica of itself. Finally, the process in the base produces data and sends them upward. Because each layer hosts at most one process for each class involving vertical data transfers, process interaction is specified unambiguously by read and write operations on ''anonymous ports.'' That is, no special management of high-level buffers is required, and the programmer easily expresses data communications as read from children (versus write to parent) operations in the high-level language available.

The main disadvantage of this scheme is the context switch mechanism. This involves saving a fairly large amount of data at a very low level of detail: status of PEs, status of vertical communication buffers, etc. The worst aspect is that the switch can slow down the pyramid in its vertical transmission capa-

bilities. For this reason such a proposal has been discarded in the SPHINX system. It remains a very interesting (and elegant) solution for a general control subsystem of a linear arrangement of processing units, provided that the context switch adds negligible overhead; it is likely that coarse-granularity tasks (rather than bit-serial ones) could benefit from it.

9.2.2.3. Generalized Control through Named Channels

The communication protocols embedded by the two control subsystems described in the previous sections handle the synchronization of data exchanges at the bit level, using the registers of the PEs in SPHINX as the shared resource. Such a fine-grained synchronization requires very high speed on the low-level controllers.

A more general approach[24, 25] consists of considering the planes of the pyramid as general-purpose processors, tightly interconnected through a shared memory. In the SPHINX system, this memory is the external memory of the PEs, which in the design of the pyramid insists on the vertical communication links and is realized as a dual-ported memory.

The control subsystem is thus organized around the concept of "message passing." The plane controllers interact through statically allocated *channels*. Channels are physically created as reserved areas of a given length in the shared memory: the length can be chosen to suit the high-level granularity of communication rather than the synchronization requirements. Channels are explicitly declared by the tasks and are managed by high-level primitives such as channel allocation–deallocation, channel status read, channel inquiries. They allow for unidirectional communication; two tasks on adjacent layers therefore require at least two channels to exchange data. Nothing prevents tasks from creating more channels of the same type.

Channel access obeys as usual a producer–consumer protocol: the shared area is guarded by an "indicator flag," whose integrity is maintained through a proper hardware mutual exclusion mechanism. It consists of synchronizing memory accesses from adjacent layers on opposite phases of the system clock cycle: if plane p accesses the dual-ported memory on the high phase of the clock cycle, plane $p+1$ uses the low phase. Thus, the guard of a channel can never be updated concurrently by two tasks on different planes, and this allows us to code the basic primitives of the producer–consumer protocol with an automatic protection scheme.

Process synchronization through named channels allows considerable more flexibility than the other schemes. The programming environment can be based on the object paradigm. Data communication can be handled with high-level

languages with no regard to the bit-serial processing typical of the hardware. The low-level controllers automatically take care of bit streams with negligible overhead: synchronization at bit level is carried out only on accessing the guard of a channel, not on all the bits of the message being written in the channel. The compulsory use of explicit naming of channels does not impair the clarity of programs. The throughput of the system can be increased by creating both blocking and nonblocking synchronization primitives. Moreover, the dual-ported memory not only allows a speedy synchronization but also helps in augmenting throughput by letting processes deposit messages and continue to work. Although optimized for a hierarchical (linear) system, this multi-SIMD control modality has many of the characteristics of MIMD shared-memory systems and is the most general conceived for pyramid machines.

9.2.3. Multi-SIMD Control Across Space

It is typical of heterogeneous hierarchical systems, such as those described in Chapter 8, to adopt a variety of control strategies. Generally, a richer environment is possible with respect to the fine-grained control of ascending and descending data movements that is common with the multi-SIMD across-resolution mode.

Heterogeneous systems support a form of MIMD processing among the levels of the hierarchy. In examining the requirements of a general-purpose image understanding parallel system, Weems[26] highlights some control strategies possible when the hierarchy of the system matches the levels into which image understanding is usually broken down. He advocates the need for a single ''thread of control'' at the lowest levels, eventually with local branches due to different associations of pixel data. At the higher levels, he suggests the need for a predominant task parallelism—that is, the complement of ''subsidiary data parallelism'' at the iconic level.

The use of a hierarchy of heterogeneous processing layers differs in upward and downward directions. The former supports data reduction and region merging and is therefore data driven; the latter supports the realization of alternative processing strategies (goal driven) and is a flow of control rather than a flow of data. Typically, a single task is instantiated at the highest level of control (the host) and migrates downward by splitting and replicating, thus originating different subtasks. The interaction among such subtasks can be the most general: loose global synchronization, tight cooperation, or even completely asynchronous concurrence.

A special case of such alternative control strategies is the one that assigns different tasks to different regions of the images. The architecture that best

supports it is the partitioned, heterogeneous pyramid of clusters. An instance of such a system is the Warwick Pyramid System described in Section 8.2.

The control system of a set of clusters is a distributed environment.[27] Each cluster is itself a small, hierarchical heterogeneous subsystem. The subarray of serial processing elements is driven in SIMD mode by a dedicated, bit-slice sequencer. The transputer on top of the cluster sends to the sequencer the macrocommands to be translated and delivered to the subarray. Multiple processes can be active in the transputer: beyond communications with other transputers, they can effectively trigger more tasks for the SIMD subarray, thereby creating a time-sharing multiprogram environment. Such vertical communications use a shared memory as the common resource.

When the Warwick system consists of more clusters, the bidimensional MIMD network thus obtained can run both completely independent tasks (the subarray edges are connected in local toruses) and cooperating ones. If cooperation is at the symbolic level, the usual synchronization mechanisms of the transputers are used. Instead, when all the clusters or a part of them want to enter a tightly orchestrated processing mode, a special hard-wired synchronization mechanism is activated from the sequencers. A network of wired-OR signals is used to create a barrier for process synchronization: when all clusters meet the barrier, the pyramid can enter a strict SIMD mode or a more complex SPMD mode. Thus, low-level, massive data exchanges are not synchronized between different tasks where each one has control of the subarrays (this was the approach in the Multi-SIMD across-resolution mode). The adopted control strategy is considerably simpler.

9.3. CONCLUSIONS

The programming environment of hierarchical systems has been discussed and classified on the basis of the few proposals that explicitly address these systems. Up to now, these proposals never became fully developed and integrated tools because pyramid systems have not yet reached the stage of extensive industrial exploitation and the prototypes still remain confined to small- and medium-scale implementations.

Even though formalization of the hierarchical cellular logic was established 10 years ago, the languages, here described, based on extension of the C languages have not been sufficiently standardized either in the typing mechanism or in the semantics of the control structures.

The control environment is even at an earlier stage. In fact, the great variety of solutions for hierarchical architectures (fine-grained, coarse-grained,

and heterogeneous) prevents adoption of a unified scheme. The status of the art here introduced covers the basic issues of the control strategies for the most popular systems.

REFERENCES

1. W. D. Hillis and G. L. Steele Jr., Data Parallel Algorithms, *Comm. ACM* **29**(12), 1170–1183 (1986).
2. K. E. Iverson, *A Programming Language,* Wiley, New York (1962).
3. C. A. R. Hoare, Communicating sequential processes, *Comm. ACM* **21**(8), 666–677 (1978).
4. J. M. Sipelstein and G. E. Blelloch, Collection-oriented languages, *Proc. IEEE* **79**(4), 504–523 (1991).
5. J. Mehat, La programmation des machines SIMD avec le langage de programmation Pyr-e, BIGRE, September, 1989.
6. A. P. Reeves, Parallel Pascal: an extended Pascal for parallel computers, *J. Parallel Distrib. Comput.* **1,** 64–80 (1984).
7. *C* Programming Guide,* Thinking Machine Corporation (1991).
8. G. Sabot, *The Paralation Model: Architecture-Independent Parallel Programming,* MIT Press, Cambridge, MA (1988).
9. S. Chatterjee, G. E. Blelloch, and M. Zagha, Scans primitives for vector computers, Proc. Supercomputing 90, November, 1990.
10. D. Nassimi and S. Sahni, Data broadcasting in SIMD computers, *IEEE Trans. Comput.* **TC-30**(2), 101–106 (1981).
11. J. Levaire and L. Bougé, On the semantics of massively parallel SIMD control structures, Report N. 91-06, LIP, Ecole Normale Supérieure de Lyon (1991).
12. G. Plotkin, An operational semantics for CSP, in *Formal Description of Programming Concepts* (D. Bjorner, ed.), IFIP TC-2 Working Conf., Garmish-Partenkirchen (1982).
13. P. Clermont, A. Merigot, and B. Zavidovique, Programmation et contrôle des machines hyperparalléles, Lecture Notes Ecole Internationale d'Informatique de l'AFCET, Friburg, Switzerland, 1990.
14. S. L. Tanimoto, A hierarchical cellular logic for pyramid computers, *J. Parallel Distrib. Comput.* **1,** 105–132 (1984).
15. S. L. Tanimoto, J. P. Crettez, and J. C. Simon, Alternative hierarchies for cellular logic, Proc. 7th Int. Conf. Pattern Recognition, Montreal, Canada, 1984, pp. 236–239.
16. J. Serra, *Image Analysis and Mathematical Morphology,* Academic Press, New York (1982).
17. J. J. Pfeiffer, HCL: a language for low-level image analysis, *J. Parallel Distrib. Comput.* **8,** 231–244 (1990).
18. V. Di Gesù, A high level language for pyramidal architectures, in *Pyramidal Systems for Computer Vision* (V. Cantoni and S. Levialdi, eds.), pp. 329–339, Springer-Verlag, Berlin (1986).
19. V. Di Gesù, B. Lenzitti, and D. Tegolo, A medium level language for pyramid machines, in *Issues on Machine Vision* (G. Pieroni, ed.), pp. 117–133, Springer-Verlag, Wien (1989).
20. V. Di Gesù, "Pictorial languages: concepts and data structures, *JIETE on Pattern Recognition,* **37** (56): 520–540 (1991).
21. S. Levialdi, Software issues for machine vision, in *Issues on Machine Vision* (G. Pieroni, ed.), pp. 177–208, Springer-Verlag, Wien (1989).

22. P. Clermont and A. Mérigot, Real time synchronization in a multi-SIMD massively parallel machine, *Proc. Workshop CAPAMI*, Seattle, WA, 1987, pp. 131–136.
23. J. Méhat and A. Mérigot, Controlling and programming the SPHINX multi-SIMD pyramid machine, Proc. 2nd Symp. Frontiers of Massively Parallel Computation, Fairfax, VA, 1988, pp. 423–428.
24. S. Bouaziz, E. Pissaloux, A. Mérigot, and F. Devos, A communication mechanism and its implementation in the multi-SIMD massively parallel machine SPHINX, *Microprocess. Microprogram.* **32,** 39–44 (1991).
25. Yang Ni, Réalisation d'un circuit VSLI multi-processeur et définition d'un modèle de controle dynamique, These, Universite de Paris-Sud, Orsay (1990).
26. C. C. Weems, Architectural requirements of image understanding with respect to parallel processing, *IEEE Proc.* **79**(4), 537–547 (1991).
27. G. R. Nudd, D. J. Kerbyson, T. J. Atherton, N. D. Francis, R. A. Packwood, and G. J. B. Vaudin, A massively parallel heterogeneous VLSI architecture for MSIMD processing, in *Algorithms and Parallel VLSI Architectures* (F. Deprettere and A. Van der Veen, eds.), (1991), pp. 463–472, Elsevier, Amsterdam.

Chapter 10

Pyramidal Tools and Applications

The pyramidal architecture supports several powerful paradigms for "machine algorithms," including pyramid building, tree search, and horizontal and vertical propagation. Such paradigms are supported by a set of basic algorithms, for which many types of pyramid or pyramid-related approaches and data structures have been proposed. This chapter reviews some of these fundamental techniques from a computational theoretical point of view and in the light of their effectiveness in application development.

10.1 INTRODUCTION

This chapter gives an overview of the most important applications of the pyramid multiresolution representation. It is not an exhaustive review of multiresolution algorithms, because the number of proposals and of solutions is continuously increasing. The books edited by Tanimoto and Klinger,[1] Rosenfeld,[2] Cantoni and Levialdi,[3] and Uhr[4] (see, in particular, the paper by Dyer[5]) offer a broad perspective on multiresolution strategies and algorithms. The bibliography collected each year by Rosenfeld[6] is an invaluable source for up-to-date research on single items.

We have focused on algorithms for the compact, usually SIMD pyramid, and for the multiresolution pipeline pyramid systems because of the effort put in by research teams building the prototypes to show effective application for

their systems. The model of a pyramid computer that basically resembles the systems just mentioned has been considered for a very long time as a promising platform for solving many vision problems with logarithmic speedup over planar meshes.

We begin the chapter by quickly reviewing theoretical results that indicate the extent to which the gain of the pyramid computer is actually logarithmic. The remaining sections cover the more relevant problems of computer vision, basically at low and intermediate levels. To begin with, we have discussed irregular pyramid data structures that, even if not directly matched to the pyramid computer, are closer to it than to any other parallel platform. It can be proved[7] that for some of these alternative constructions, such as the so-called overlapped pyramids commonly used in many segmentation tasks, there are quite efficient implementations on that basic architecture.

The rest of the chapter covers segmentation, texture analysis, template matching, curves, shapes, image compression, motion, and stereo. These basic paradigms are the tools for assembling significant real-life applications: the nearly logarithmic complexity of such algorithms in the pyramidal environment and the (eventually available) true parallel pyramidal systems are the pathway for substantial speedups and increased system throughput.

10.2. COMPLEXITY OF SOME BASIC ALGORITHMS

The rich and regular interconnection structure embedded in the basic quad pyramid (see Chapter 3 for a formal definition and Chapter 5 for examples of machines that implement this topology) has stimulated great and persistent interest in the capabilities of such a network to support fundamental algorithms drawn from areas such as computational geometry, graph theory, and statistics. With respect to computer vision and image processing, such algorithms are important from the theoretical point of view, because they form the basis for a wide computational platform, and from a more practical point of view, since many of them can become part of the software libraries of functioning pyramidal machines.

Since its introduction, the pyramid computer (PC; in the following, we refer to the standard quad-pyramid computer of base edge length $N = 2^n$) has been proposed as an efficient parallel computer architecture because it merges the structure of an $N \times N$ standard mesh with that of an $(n+1)$-level quad tree. The logarithmic diameter of the PC due to the tree component is the basis for obtaining logarithmic time algorithms not achievable in the mesh. While this is indeed an established result for some basic problems (bit counting, area, perim-

eter, and in general all algorithms in which a processor receives a constant amount of information from its children), a series of works by Miller and Stout[8] have proved that such a target cannot be obtained for the majority of geometry-related problems. The theoretical analysis carried out has highlighted lower bounds for worst-case problems[9] and has produced some standard data movement routines[10] specially suited to the PC for minimizing the overhead due to the tapering structure of the pyramid.

Historically, among the first contributions to basic algorithm design for the PC is the analysis of sorting, histogramming, median filtering, and other statistical operations carried out by Tanimoto.[11, 12] The sorting algorithm consists of iteratively moving the local maxima, detected at each PE, upward through the levels with four compare-exchange operations. In $\theta(\log N)$ time the first maximum is at the apex, where it is extracted and replaced by the smallest number in the representation used. Successive maxima emerge one after another in constant steps. This approach to sorting is not the best from a purely theoretical point of view, since it has $O(N^2)$ complexity, but it has the advantage of being suited to the serial extraction of the sorted list out of the apex of the pyramid; moreover, it can be used for median filtering. Indeed, at a given intermediate level in the PC, the sorting algorithm produces the series of maxima contained in the underlying subpyramids, and on the basis of this Tanimoto builds the median over chosen windows by iterating the local sorting operations.

A theoretical result by Stout[13] shows that the PC is inherently equivalent to the mesh as a sorting machine. The well-known $\Omega(N)$ limit in the mesh cannot be improved through the pyramid's upper layers. Sorting requires moving $\theta(N^2)$ data from one half of the base to the other half, and the maximum bandwidth available is the number N of links between the two halves, so the required time is $\Omega(N)$.

Moving data between PEs in the base, or even between PEs of different levels, is a fundamental operation in many geometric problems. An extensive study of pyramidal data movements[10] has highlighted the central role played by the midlevel of the pyramid, which contains $\sqrt{N} \times \sqrt{N}$ PEs. This level was shown to be the bottleneck for those data movements that require a massive exchange of data between distant areas in the base of the pyramid. As an example, algorithms that adopt a divide-and-conquer strategy[14] are best implemented as follows: in a first phase, the $O(N^2)$ data are reduced to $O(N)$; these are moved quickly, through one of the specialized pyramid move routines, to the middle level of the pyramid, where they are shifted in meshlike mode; the downward movement is then carried out with another specialized technique, known as *funnel read*.[10]

The connected component labeling problem is solved in this way with an asymptotic complexity $O(\sqrt{N})$, a substantial improvement over the $O(N)$ result obtained in the mesh. Furthermore, this strategy is much more general than the approach followed by Tanimoto:[15] "compact" objects are labeled in $O(N)$ by using only the quad-tree connections.

A host of optimal algorithms for computational geometry problems is solved by Miller and Stout.[16] They also show that many algorithms studied for a single geometric object remain valid when the base of the pyramid contains more figures. They introduce the notion of "essential pyramid",[17] which simulates a true pyramid whose base covers a single figure in the scene, thus allowing the use of algorithms designed for complete pyramids with a single object.

10.3. SPECIAL PYRAMIDS

The basic architecture of the quad pyramid is not the only possible hierarchical structure for exploiting logarithmic grouping properties. In literature, other pyramidal arrangements have been proposed that try to solve the discontinuities introduced by the rigid subdivision of the space into nonoverlapped quadrants generated by the basic pyramid.

Essentially, we can identify two broad classes: regularly sampled, *overlapped* pyramids, and *irregularly* sampled, *special* pyramids. The former class includes the "dual pyramids"; the latter includes the "stochastic," "adaptive," and "custom-made" pyramids.

10.3.1. Overlapped and Dual Pyramids

Overlapped pyramids are built by using a reducing factor r smaller than the support w that links a parent to its children (see Section 3.5.4 for a formal definition). The sets of elements in the base of the pyramid that are descendants of a given node in an upper level are known as the *receptive field* of that node, in analogy with the structure of the peripheral vision system in humans. A very common situation is the case in which a parent has 16 children ($w = 16$) arranged as a 4×4 block and a child has four possible parents, reflecting the fact that 4×4 blocks overlap horizontally and vertically by 50%; the resulting reduction factor r is 4. Variations of this case are possible, and a systematic classification is available in Kropatsch.[18] This structure has been widely used in segmentation tasks, as we will see.

Another form of overlapping is that generated by the "dual pyramid."[19]

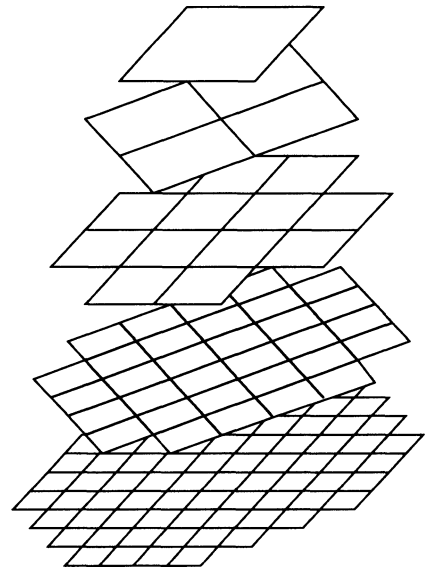


Figure 10.1. The construction of a "dual pyramid." Alternate tessellation directions at 45° produce a 50% overlapping.

In this case, a recursive method generates a pyramid by rotating the meshes by 45° at alternate levels. This is obtained by considering half the diagonal of a square as the edge of the square of the next level; this produces a degree of overlapping equal to 50%. The top-down generation of the dual pyramid proceeds by placing a 2×2 square over the center of the parent; the length of the edge of each subsquare is half the diagonal of the parent. The lower levels as a whole soon take the shape of an octagon (see Figure 10.1). The resulting pyramid has a variable resolution, a variable reduction factor, and a fixed support $w = 4$. Thus, each node has a maximum of two parents and exactly four children, located at equal distances from its center. This last feature is notably different from the other overlapping cases mentioned and could be helpful during segmentation by dynamic linking, as we shall see in Section 10.4.

10.3.2. Stochastic, Adaptive, and Custom-Made Pyramids

The second class of pyramids, *irregular pyramids*, alters the structure of the basic quad pyramid quite drastically. A first irregular component is introduced in the decimation process, which reduces the nodes of the successive levels. Instead of applying a regular sampling grid, as in all geometric pyramid construction, Meer[20] has proposed a random selection process, whose outcome

is the ''stochastic pyramid.'' In this pyramid, the nodes of the level above the base (and recursively for the upper levels) are chosen from those in the base by applying a maximum criterion to the outcome of a stochastic process, which is executed in all base cells. A cell survives and becomes a node in the level above only if the randomly generated value is the largest among those of its immediate neighbors (in truth, the random process is used three times to obtain a stable configuration). The average reduction factor (decimation factor) is 5.44, with a negligible standard deviation. The selection of the nodes in the upper levels is only half of the process in generating the hierarchy. The second half consists of establishing hierarchical links between parents and children. This process is started from the surviving cells (parents) and is actually an expansion in the neighborhood. An expanding parent ''conquers'' more and more cells that make up its support. The growing support of a parent eventually touches the growing support of another parent; in this situation the expansion stops, and the shared child links itself only to the parent having the strongest random variable. The stochastic pyramid can also be defined in the monodimensional case with straightforward simplifications.

The resulting hierarchy is very irregular, though it has been used efficiently in some algorithms as an alternative to the more usual pyramid, such as curve representation,[21] bimodality analysis and object–background delineation,[22] connected component analysis of labeled images, and segmentation of gray-level images.[23] The advantages of this approach are essentially the shift-invariance property of the stochastic pyramid, whose structure does not depend on the geometry of the plane tessellation, and its robustness to structural perturbations.[22]

The random election process of the stochastic pyramid has no relation at all to the content of the image in the base of the pyramid. To obtain a more realistic hierarchy, Jolion and Montanvert[24] introduced the ''adaptive pyramid.'' In such a pyramid, the decimation process is carried out by letting cells survive if they are a local extreme of an *interest operator*. For example, in a segmentation task it can be useful to isolate regions of pixels with a similar gray level. The chosen operator in this case is the variance of gray level in the region, and the surviving cells are those where the variance is a minimum. The growth of the regions (that is, the identification of the children and of the receptive fields) obeys an expansion rule similar to that of the stochastic pyramid. The links are established with the parent according to a criterion on the gray-level contrast: the least contrasted cell in the support is the best parent and therefore is chosen. The adaptive pyramid is slightly larger than its stochastic counterpart and was used successfully on the same type of problems with comparable results.

A spectrum of pyramid representations that bridges the regularly sampled ones with the more irregular, adaptive pyramid was proposed by Peleg *et al.*[25] They used the term ''custom-made'' pyramid to denote a type of geometric construction that resamples a regular $M \times N$ grid into a regular $K \times L$ smaller one; the resampling can be uniform or weighted, and in both cases a cell in one level contributes a finite quantity to its ''parent(s)'' in the level above. The construction of the pyramid results from the iterative application of the resampling scheme at successive levels, possibly varying the geometry of the next grid at every next level. With respect to overlapped pyramids, the hierarchy is more variable, but it is still determined by the geometric parameters defining the resampling ratio. However, in this structure, too, the resampling can be made dependent on the image content by using an interest operator, which influences the weights. Peleg *et al.* used a measure of the ''busyness'' of a region to establish the variable degree of resampling: a smoothed absolute value of the Laplacian gives a clue to the regularity of the region. The coarser the region, the wider the support used in building the next level; areas richer in contrast, on the contrary, are sampled more finely.

10.3.4. Centralization Graphs

A general mechanism for implementing interresolution communications in pyramids has been proposed by Clermont.[26] It is based on a graph that has a subset of the pyramidal topology defined in Section 3.5.4. This graph is called a centralization graph, or C-graph, and consists of a subset of the pyramid links selected on the basis of the full-resolution image data (pyramid base).

Let us call R the planar graph of the base in which the centers of segment pixels that are adjacent are linked (with 4- or 8-connectivity); $F(p)$ the projection of a node p, consisting of the set of the descendent extrema of p (nodes without descendents); $F(G)$ the projection of a graph G (a general subset of the pyramid graph) obtained by the union of all $F(p)$ with p belonging to G.

A pyramid subgraph G is a C-graph if and only if

- $F(G)$ is included in the pyramid base.
- For each p belonging to G, $F(p)$ is connected.
- For each p_1 and p_2 connected in G, $F(p_1)$ and $F(p_2)$ are connected in R.

Figure 10.2 shows two instances of C-graphs for an 8×8 binary image. Instead of obtaining a unique tree, a C-graph builds up a forest. The roots of the trees in the centralizing forest are called CEN nodes.[27] In general, an image

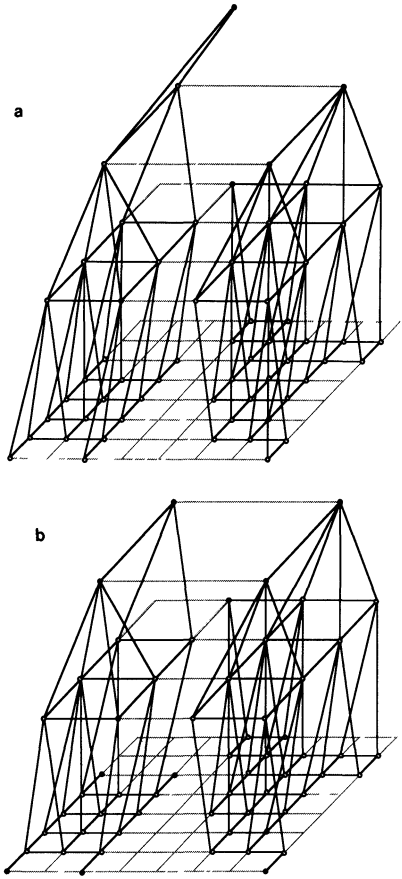


Figure 10.2. Two examples of C-graphs built on the same image containing two segments, by adopting a 4-connectivity. The CEN nodes are highlighted by ○ (ordinary nodes by •).

segment can have many CEN nodes (see Figure 10.2). An image segment is characterized by subregions with the following properties:

- Each full-resolution segment pixel belongs to one subregion.
- Every subregion is dominated by a subtree of the pyramid.
- Trees which correspond to different subregions are disjointed.

On the basis of the C-graph, global computations on a connected set can be effectively computed in parallel by reduction operations from the pyramid base to the CEN nodes, by downward broadcasting in the reverse direction, and with subregion communication through the intralevel links. These primitives supply a common framework for dealing with large and irregular data

movements on image segments working in SIMD mode on a fine-grained pyramid. Many intermediate-level image processing tasks (such as labeling, segmentation, polygonal approximation, etc.) can benefit from these primitives.[27]

10.4. PYRAMIDAL TECHNIQUES

While Chapter 2 covers very general multiresolution strategies, the following section details specific techniques useful in many applications. Common characteristics of these techniques are the overall speedup in the applications and the effective use of limited computing resources in comparison to the complexity of the tasks. Starting with the exploitation of pyramids in mathematical problems, we shall then consider classical image-related problems such as segmentation, object recognition, image coding and transmission, and stereo and motion analysis.

10.4.1. Multigrid Numerical Methods

Many image analysis problems can be formulated with a common mathematical foundation according to variational principles or to partial differential equations (PDE). Among them are optical flow computation (see Section 10.6) and shape-from-shading.

The formulation of such physical problems in the continuous domain either as the minimization of a functional or as the solution of PDE with proper boundary conditions leads, after discretization, to the solution of large systems of linear equations of the form $Ax = b$, where A is a band sparse matrix, b is the vector embedding the boundary conditions and other external terms, and x is the vector of the points to be calculated. In the domain of vision, x is itself a large grid of points that uniformly sample the physical world (surface, moving points, etc.) at a single resolution. Among the numerical methods for solving the linear systems, those based on iterative algorithms best match the computational characteristics of vision problems based on near-neighborhood primitives.[28] Unfortunately, the propagation across the grid of the approximate solution is very slow. Since nearly all the formulations of the underlying physical problems are based on a global constraint, such an extended propagation is necessary to obtain a good solution.

To alleviate this situation, mathematicians have conceived a new class of algorithms that considerably speedup the convergence by using multiple grids for the approximate solutions. Such algorithms are known as "multigrid methods." A fairly complete review of their mathematical properties can be found

in Hackbusch.[29] We will only consider instances of such methods in the vision context, based on the version of multigrid relaxation as formulated by Brandt.[30] Applications of this principle to low-level computer vision are reported by Glazer[28] and Terzopoulos[31, 32] in optical flow computation, visual surface reconstruction, shape-from-shading, and brightness determination. A detailed discussion of the applications themselves is omitted; however, optical flow is discussed in the section regarding motion.

Following Terzopoulos,[32] we can identify the following components of the multigrid methodology: (i) a multiresolution representation of the field of analysis; (ii) the local iterative process that creates the approximate solutions at each level; (iii) a coarse-to-fine and a fine-to-coarse distribution process, which propagates the approximate solutions in the hierarchy, thus effectively speeding up the convergence of the algorithm; (iv) a global strategy, which dispatches the control of algorithms in the various levels and activates the upward and downward data movements.

The multiresolution representation consists of a stack of meshes built from the finest one by straight subsampling. The reduction factor is usually 4 and the grids are aligned, so each point in a coarser resolution level is registered with a point in each of the finer ones. This multigrid structure is closely related to the definition of the basic quad pyramid. The real difference is in the data transfers between levels. Indeed, the fine-to-coarse distribution process used in this method is a straight injection from the grid point in the finer level into the corresponding point in the coarser one; moreover, the coarse-to-fine downward data movement is usually obtained through bilinear interpolation.

The local iterative process executed on the multigrid structure speeds up the computation by quickly deriving approximate good solutions at a coarse level, where the near-neighbor propagation due to the relaxation spans wide sections of the basic domain. The rationale for this fact comes from an analysis of the contributions to the errors in the approximations. At a given level in the hierarchy, the iterations of the relaxation process quickly cut off the high-frequency components of the error but cannot eliminate the low-frequency ones. However, the frequency spectrum is split into octaves from the multiresolution structure, and the octave of the next coarser level includes these low-frequency components. The local process in this coarser level takes care of approximately half the octave and recursively redirects its low-frequency half-octave to the next coarser one (the mathematical foundation of this behavior of multiresolution representation is highlighted in Chapter 2 in the discussion regarding wavelet decomposition).

The global control strategy of the multigrid approach can follow two approaches (see Brandt,[30] Glazer,[28] and Gannon[33] for details). In the former, the

upper levels execute the local relaxation process to compute the updates to the solution, which exists in full only in the finest level; the convergence criterion is a threshold set at the beginning for the finest level and is used to derive dynamic thresholds for upper levels. The second approach has an approximate solution at all levels, and the local relaxation process computes a new approximation and a new set of updates for the benefit of neighboring levels. These updates are propagated as described above. Following Gannon,[33] the flow of data can be depicted by a "V" cycle having two sequential steps, the fine-to-coarse phase preceding the coarse-to-fine one. This method is more amenable to computer vision problems since it directly computes the solution at multiple levels of resolution. The terminating conditions of the local iterative processes depend on global information, since they are usually written as a comparison between a threshold and a norm on the error. Sometimes, fixed schemes based on an *a priori* limited number of iterations are suitable as well; they keep all computations on a strictly local basis.

The multigrid approach is a major application to computer vision that benefits from the massively parallel homogeneous pyramid described in Chapter 5. The shortcomings of the method due to the homogeneous handling of the field of analysis (to be discussed later in the contest of motion) can be easily overcome on the same parallel platform.

10.4.2. Image Segmentation

The hierarchy embedded in the pyramid offered a sound computational paradigm for image segmentation and was used with the various approaches typical of this task. We can group them into region-oriented (a homogeneity criterion groups pixels into "uniform" areas) or border-oriented categories (the discontinuities in some image property, such as gray level, color, local texture, etc., are used to delineate one area from the others). In this section, we review the ways in which the hierarchy helps in the implementation of algorithms in both strategies.

10.4.2.1. Region-Based Segmentation by Pyramid Linking

The first proposal for a hierarchical segmentation based on a region property was made by Burt *et al.*[34] on the basis of an overlapped gray-level averaged pyramid. The pyramid is initially constructed by simple averaging of the 4×4 pixels that make up the support of each parent node (as noted, the degree of overlapping is 50% along the cardinal directions). The hierarchical links are then established (starting from the base) by choosing, for each pixel in the

lower levels, one of its four candidate parents on the basis of a homogeneity criterion (e.g., the minimum gray-level distance). Once this choice is made, the gray-level values in the parent level are recomputed by using only the values of the linked children. The process is then iterated until convergence is obtained. Since at any level one pixel is linked to a single parent out of four candidates, the end effect of this procedure is to segment the image into four regions. These are the ''receptive fields'' of the four nodes in the next-to-last level of the pyramid: the four trees that originate from this level are the four most homogeneous partitions of the averaged gray-level pyramid.

This basic scheme has been improved in many aspects by subsequent proposals. Hong *et al.*[35] studied a variation based on ''weighted linking'': here the hierarchical connections between two adjacent levels are computed by making each node contribute initially to all its four candidate fathers. The weight of this contribution is iteratively reinforced or decreased until a stable situation is obtained. The choice of weighting function is critical and can lead to weights that converge to 0 and 1 (as in the previous case) or to equal values. The effect of including the values of the within-level neighbors also proved to be helpful, especially when dealing with the first levels of the pyramid.

In all cases, the hierarchy of links extends upward throughout the pyramid. A different strategy has been pursued by other authors;[36, 37] it leads to a forest-based segmentation. The linking process is *unforced*. This means that averaged gray-level pixels in intermediate levels are allowed to become a local ''root,'' without linking to any of the candidate parents, on the basis of a local intralevel criteria. Non-root pixels are assigned to the strongest candidate parent, and this final top-down linking phase produces the segmentation of the image. Rooting strategies and some measure of optimality in the linking process have also been investigated.[36]

A stochastic pixel relinking method was proposed by Spann[38] to segment a figure against the background. The pyramid is a composite data structure with an overlapped 4×4 support: each node stores both gray-level values and spatial moments computed in hierarchical fashion according to Burt *et al.*[34] Children-to-parent links are either on (value 1) or off (value 0). The updating of the links is guided by a simulated annealing process that leads to global minimization of a locally based measure of fitness to a model. The resulting algorithm performs especially well in low signal-to-noise ratios.

10.4.2.2. Boundary-Based Hierarchical Segmentation

The process of delineating objects in an image by the use of discontinuities (boundaries) is the dual of the region-based one and was the first paradigm of

multiresolution planning, as described in Section 2.5. In general, the hierarchy can be exploited in two main ways. In the first approach, edges are detected at the highest resolution available and then traced, through the reduced resolution according to the scale-space methodology,[39] on the basis of the zero-crossing persistence across-resolution property.[40] In the second case, the edges are computed at multiple resolutions on averaged gray-level pyramids and then linked in two steps (bottom-up and top-down respectively[41]). Alternatively, edges are used to identify pixels in low resolution that are liable to be the ancestors of compact objects at high resolution.[42]

The former approach will not be discussed here, mainly because the majority of the properties were already introduced when we analyzed the Laplacian pyramid in Section 2.4.3.

The linking approach pursued by Hong *et al.*[41] tries to capitalize on both of the representations of the gray-scale pyramid and of the discontinuity pyramid; the latter is computed at all levels by applying a local edge detector to the nonoverlapped averaged gray-level pyramid, initially built on top of the image. The critical phase of linking edges is, however, performed by using 4×4 squares of 50% overlapped support in a bottom-up procedure. A straightforward criterion selects the parent node by comparing the local edge direction with that of the four candidate parents and then establishes the link with the most similar one. This strategy allows unforced linking when no candidate parent is within a given thresholding angle with respect to a node edge direction. Hence, many local roots appear in the pyramid. If the image contains compact objects with smooth boundaries, the algorithm is able to consistently delineate the objects, but if the shape of the boundary is rich in different curvatures the hierarchical description of its edges will contain many roots at different levels of the pyramid. A later top-down process is proposed to rank local edge segments as belonging to the same boundary.

The edge information computed at different resolutions was used in another way without explicitly linking edge segments across resolutions.[42] To guide the identification of compact objects, the pyramid of edges helps to identify locations at low resolution that are surrounded by strong edges. This cue is used in top-down mode by propagating ''scores'' indicating ''compactness'' computed in the locally delineated regions. Thus, the edge information is used consistently with the gray-level one to control top-down region-growing, region-splitting process in a bounded search.

The hierarchy of the 4×4 overlapped pyramid is used to reduce the computational complexity of an edge detector that extends the Hueckel operator in multiresolution.[43] In this proposal, the sharp circular function that Hueckel uses to model an edge in the image as the best fit to line discontinuity is replaced

by a more regular Gaussian. This choice allows us to combine the construction of the multiresolution gray-level pyramid with the convolution by Gaussian-like kernels. The key point in the algorithm, however, is the hierarchical computation of the approximation of the edge model at multiple resolutions. The approximation consists of the projection of the image over the functions that generalize Hueckel's basis. In this case, such functions are polynomials (of degree 2 at the most) of the pixel coordinates. Therefore, the projections are easily computed as linear combinations of moments; and moments can be obtained very efficiently on the pyramid.[34, 44]

It is well known that noise is a major problem when locally based edge detectors are used. This problem is partially removed by acting on a smoothed version of the image. We note that the overlapped, averaged pyramid used in so many algorithms for multiresolution segmentation introduces a first form of smoothing. A more precise approach was investigated by Park and Meer.[45] They use a nonoverlapped standard image to quickly compute global estimates of the noisy image. Then they implement an adaptive smoothing algorithm which classifies pixels as centers of maximal homogeneous subareas and assigns them the mean gray value of the maximal region. Pixels close to a discontinuity must be sensible to gray-level variations and must therefore receive a mean value computed on smaller subareas. The pyramid multiresolution representation is used to identify such maximal subareas. The process is top-down: pixels are ''colored'' with the previously computed mean values if the standard deviation in the two successive higher levels allows them to be considered part of a homogeneous region. Otherwise they are processed more locally with a further run of the adaptive least-squares smoothing algorithm. The end result of this smoothing technique is that edges are blurred much less than in overlapped image segmentation approaches.

10.4.2.3. Object Delineation

A special case of image segmentation is object delineation—that is, the isolation of a single object on a noisy background. The previous segmentation methods, based on pixels linking and border extraction, did not use the important property, valid in this case, that the pyramid tapers the dimension of the object in successive levels until the object degenerates to a single point. If the object is bound by a square of edge length 2^L (we assume that the shape is roughly compact), its representation in a 2×2 nonoverlapped pyramid becomes a single point at a level $L-1$ planes above that of the image.

This fact was recognized and first applied in a algorithm by Rosenfeld and Sher.[46] On an intensity pyramid built without overlapping, they applied Lapla-

cian spot detectors to identify the root of a tree having the object as its basis. The rationale for the use of the Laplacian is that the contrast is the highest when the object degenerates into a single point against a roughly constant (averaged) background. The actual identification of the root was carried out by considering three consecutive levels. The chosen root is the starting point of a tree-growing process, which uses a wider 4×4 support: every pixel compares its value with that of its four possible parents. The pixel is added to the expanding tree if it is near (using a distance that takes into account gray-level difference and spatial position with respect to the candidate parent) the parent already belonging to the same tree. When the tree-growing process reaches the base of the pyramid, pixels in the tree make up the object, while the others are in the background.

This algorithm was successfully adapted to the problem of delineating a cluster of points[47] rather than a roughly continuous gray-level surface. The tapering of the cluster across the resolution is as reliable as that involving a compact object, but the top-down tree-growing process fails to correctly delineate the pixels in the base of the pyramid. Indeed, according to the cluster characteristic, the probability even in the cluster area that a point randomly selected belongs to the pattern is quite low. Therefore, the final stages of tree growth fail to collect many such points that are much closer (according to the tree-growing distance criterion) to the background. The problem was tackled by using an instance of the general class of ''consensus algorithms,'' which seek a robust solution to a single problem by using more algorithms or more executions of the same algorithms run with slightly varied parameters.[48] In the cluster delineation problem, consensus is obtained by shifting the initial cluster by a few pixels in the base, running the delineation algorithm described above, and ORing successive reconstructions obtained in the base. The improvements are quite effective.

An important difficulty with this approach is the too sharp tapering of the nonoverlapped pyramid. A subsequent proposal[49] applies the usual 4×4 overlapped pyramid construction and then starts a top-down delineation process at a coarser resolution level. Instead of looking for the single point that summarizes the object, the process reduces the object to a set of edge pixels. For each edge pixel, two roots are created, one *interior* to the object, the other *exterior* to it. Both are used in a tree-growing process, which uses a confidence measure to assess the degree with which a pixel is classified as belonging to either class.

This is necessary because pixels located at higher levels summarize information from wide areas in the image. Being on the interior or on the exterior of the object is not a firm criterion as in the base. The use of overlapped pyramids in the construction of the averaged intensity pyramid, the confidence

measure in the tree-growing phase, and the two-tree approach (interior and exterior) considerably improve the delineation of the object, especially in high signal-to-noise ratios.

The delineation process was also carried out reliably with the irregular hierarchy defined by the adaptive pyramid.[24]

10.4.2.4. Segmentation by Texture

The analysis, classification, and segmentation of textured images was studied in depth in the context of multiresolution. This is not surprising, because the nature of texture is intrinsically related to multiple scales, in which the tonal primitives and the spatial primitives are best highlighted. In fact, it is well known that the distribution of patches of roughly constant gray-level areas can be perceived as a predominant gray-level phenomenon when the resolution is coarse. Dually, by analyzing the texture at a finer scale, the spatial structure of the patches is more evident than the gray-level content of the patches themselves.[50]

Accordingly, the analysis of texture in the pyramid environment was addressed both as the segmentation of the image into patches of uniform gray-level properties and as the grouping of structural features (texels) similar to each other and spread into areas perceived as differently textured regions either because of texel differences or because of the composition differences.

The first approach was pursued within the context of overlapped pyramids.[51, 52]. In particular, Pietikäinen and Rosenfeld[51] address the segmentation problem by initially decomposing the image into 8×8 squares of pixels. Such squares are characterized with one of the many textural features computed on the gray-level cooccurrence matrix. In this case they use the ''contrast,'' that is, the moment of inertia of the cooccurrence matrix along its main diagonal. This choice is only one of the many possible, and it is immaterial to the structure of the algorithm. Such attributed 8×8 patches are then smoothed with a median filter and used to build an overlapped, unweighted pyramid with 4×4 support. The segmentation is obtained by a forced linking process, where each cell in the pyramid chooses one among its four candidate parents. In this algorithm, however, the choice was constrained within only two possible parents, to produce a segmentation in at most two regions. An improved technique consists of first performing a rough image segmentation. An average pyramid is linked in top-down mode according to gray-level similarity. The segmentation in the base is the starting point for the upward texture linking. The authors report better performance with respect to the straight bottom-up linking pro-

cess, and argue that global information is essential in the process of texture characterization.

Kjell and Dyer[53, 54] instead follow a method predominately based on higher-level tokens both for texture analysis and for texture segmentation. They characterize texture as the grouping of elongated edge segments. To begin discrimination of differently textured image regions, they compute "texture separation maps," that is, histograms of local distributions of oriented edge segments. Such histograms are edge centered, and the *intrinsic image* which is input to the actual segmentation process has one such histogram at each pixel belonging to an edge segment. After a smoothing step (a Gaussian filtering in this intrinsic image), they perform a bottom-up pyramid forced linking process. The linking is executed on averaged local histograms, and the most similar parent node is the one with the smallest average distance computed on the properties in the histograms. Since the image is supposed to contain two regions at the most, the forced linking is restricted to two possible parents.

Other approaches to texture characterization in multiresolution use models different from the dual spatial–gray-level distribution we have just considered. The mathematical notion of fractal—a shape that is self-similar at all scales—introduces a number of invariance properties that, although only approximate in nonfractal objects, nevertheless can be used to characterize texture. A "fractal signature" for texture was introduced by Peleg *et al.*[55] It is based on the measurement of the area of the gray-level surface at different scales of analysis as suggested by Mandelbrot.[56] This parameter was shown to be extremely efficient for texture segmentation. The method applies multiresolution only when approximating the different fractal scales at which the signature is computed.

A statistical model for texture is applied by Bouman and Liu.[57] In their method, regions with different statistical properties are outlined in a top-down segmentation process. Texture is modeled as a Gaussian autoregressive field. The hierarchical structure of a nonoverlapped quad pyramid is exploited to guide the segmentation process. At each level of analysis, image classification is approached as a statistical estimation problem. The results of classification at one level are used as the starting condition for the classification step at the level below, where they are replicated in each 2×2 block according to the quad-tree topology. The method is efficient because the constraints to the classification process propagate quickly at coarse level. Its relationship to multigrid techniques[32] is quite evident. The authors maintain that the multiresolution approach, in this case, not only decreases computations (a result analogous to that of multigrid methods) but also improves segmentation quality.

10.4.2.5. General Assessment of Segmentation Techniques

A critical analysis of pyramid segmentation algorithms was carried out by Bister *et al.*[58] They addressed the robustness of such algorithms with respect to shift, rotation, and scaling of the input image, and show, through a thorough set of experiments, that the majority of algorithms is indeed variant to these transformations. The reason for this sensitivity lies with the subsampling operation which is central to all pyramid construction algorithms. While any single image can be safely handled by a properly tuned segmentation scheme, no scheme is able to correctly process even slightly changed copies of the original image. An especially critical case is that of elongated objects. When analyzed in the spatial frequency domain, such objects exhibit two fundamental sets of frequency components, one along the transversal section (width, associated with higher frequencies), the other along the longitudinal section (length, associated with lower frequencies). The pyramid structure itself considerably tapers off the higher components; this is even truer when Gaussian convolutions are used in the pyramid-building process.

The analysis of pyramid segmentation algorithms has highlighted the already cited higher robustness of schemes based on stochastic pyramids. All the same, the irregular structure of such pyramids prevents a fair comparison of segmentation results, since the stochastic linking process produces quite different pyramids out of slightly changed input images.

10.4.3. General Matching

The advantages of hierarchical multiresolution in coarse-to-fine search strategies were discussed in Chapter 2, particularly Section 2.3.3. The features looked for are first searched at a coarse resolution: successive refinements limit the search space, by consistently guiding the process where there is enough evidence of a good match, and increase the precision of localization with adjustments of the initial guess.

Many algorithms have been implemented according to this strategy. Dyer[5] categorizes them on the basis of the use of the refinement strategy which can be applied to a single image, in a general template-matching algorithm, or to couples of images for solving many types of common correspondence problems (e.g., stereo and motion).

We briefly describe one algorithm among the many possible, with the sole purpose of highlighting the strategy common to all of them. It is the classical template-matching problem that has been addressed by many authors in the early period of multiresolution research activity.[59–62]

Tanimoto[61] analyzes the possible strategies for implementing a template-matching operation on the pyramid. Basically, he uses a quad pyramid to apply variations of a recursive descent policy. The correlation-matching operation, on a small template window, produces approximate results in a coarse resolution image. When a sufficient match is found at a cell in the coarse level, the procedure calls itself recursively to examine the descendents of this cell. The simplest descent policy follows the quad-tree structure, by examining in turn the four descendents of each node, until a predefined number of levels has been examined or the base of the pyramid is reached. In descending the pyramid, successive figures of merit of the matching are appended together. If a given path through the quad tree does not produce a good result, the procedure backtracks and examines another path. Tanimoto studies alternatives to this ''all-of-four'' descent policy. He proposes an ''all-of-sixteen'' strategy in order to broaden the search and to diminish false negative detection (note that the overlapping introduced in this case is only the partial superposition of search areas; it is not related to the averaged overlapped pyramid construction used in segmentation by linking) and ''best-of-four'' and ''best-of-sixteen'' strategies to avoid backtracking and to limit the search at each new level by choosing the single child where the correlation match is highest. Other parameters considered are the choice of the threshold that discriminates between good and bad matches: it can be fixed throughout the levels of the pyramid or decrease with each level. The analysis of results shows that the use of an overlapped 4×4 support in the ''best-of-sixteen'' recursive descent allows us to reliably locate all true good matches at high resolution. However, it is not optimal from the computational point of view because a cell can belong to more paths that traverse the pyramid downward, because of the overlapping 4×4 support.

10.4.4. Lines, Curves, and Shapes: Description and Recognition

The hierarchical structure of the pyramid has been used for studying shapes by mainly following two approaches. In the structural descriptions of a shape the hierarchy helps in aggregating consensus during a form of ''election'' or ''voting'' process: roughly the same detection is carried out on each level of the pyramid, but on a more compact intermediate representation of the shape. Alternatively, the multiresolution capability of the pyramid is used to built various versions of the shape, and the emphasis is more on regularizing the outline of the curve rather than on explicitly detecting the curve against a noisy background.

We shall give a short summary of the algorithms proposed for shape detec-

tion (starting from lines) and for shape, with a brief hint regarding the concept of "hierarchical symmetry."

10.4.4.1. Lines

Detecting lines is a very important and much-studied problem. One of the most common techniques is the Hough transform,[63] which is a structural way of characterizing lines and shapes analytically. Shapes whose model is not given in analytical form, e.g., it is given by points (templates), can be analyzed through the generalized version of this transform.[64]

Shapes characterized by analytical models are detected starting from the pixels of the image through a voting process, whereby each point in the image chooses the possible values of the parametric representation of shape for which it is compatible. In the case of lines represented by the couple (ρ,θ), e.g., distance from the origin and its orientation with respect to the x axis, a point (x_p,y_p) votes for all (ρ,θ) in which $x_p \cos(\theta)+y_p \sin(\theta)=\rho$. The set of values for the parameters is the "parameter space." The required quantization of the parameters usually makes this space rather large, and this is one of the implementation complexities of the method.

A hierarchical approach to the Hough transform by Li *et al.*[65] addresses exactly this problem. The method is not specific to the pyramid architecture but is hierarchical in nature, because it uses a hierarchical representation of the parameter space. It consists of allocating space dynamically where the voting process is stronger, without explicitly representing all possible values of the parameters. The quantization is variable and adjustable to the situation. The approach is interesting, but only covers the detection of a single line.

A pyramid Hough transform for lines was proposed by many authors. Tanimoto's algorithm[66] aims not so much at the construction of the parameter space, with peaks identifying straight lines, but explicitly detects all instances of straight lines in their parametric representation (ρ, θ) by using a hierarchical "election strategy." This strategy builds evidence of the instances by merging, at decreasing resolution, those contributions from the hierarchical neighborhood that are likely to correspond to the same line. To do so, it uses a similarity measure based on the differences of the ρ's and θ's. A sketch of the algorithm follows.

The base of the pyramid is supposed to be large enough to store the whole image in the $n \times n$ mesh. The parameter space is not explicitly mapped anywhere in the pyramid, since the algorithm only produces a list of m couples (ρ, θ) representing the m most evident instances of leins in the base. The computation carried out on the pyramid proceeds bottom-up, starting from the base,

and consists of two phases. The former only involves the base of the pyramid, the latter uses the upper layers to merge evidence of the lines.

In the first phase, the image is processed with an edge detector operator, and each pixel is assigned a feature vector (θ, γ) describing the orientation θ and the strength γ of the edge across it. Then the hypothesized line in the base (level L) with direction θ passing through pixel (i, j, L) is identified with its distance ρ from the origin of the coordinate system on the base mesh. The local evidence for such a straight line is represented by the vector (θ, ρ, γ).

The second phase merges local evidence for lines into clusters of roughly "similar" contributions. The similarity function measures the distance between contributions p and q by weighting the absolute differences $\theta_p - \theta_q$ and $\rho_p - \rho_q$. The four clusters associated with the children of pixel (i, j, l) are compared and merged if they stay within a chosen distance from one another. Then only the first m contributions (θ, ρ, γ) are retained and the others are discarded. This merging process is repeated at each new level until the apex is reached; the m contributions that have survived are the sought for m straight lines.

A very similar "election" strategy is followed by Jolion and Rosenfeld.[67] Collinear straight segments close to each other are merged together and ranked on the basis of their length.

The method of Princen *et al.*[68] adopts an overlapped pyramid. The image is subdivided into overlapping subsquares, and a preliminary Hough voting process is executed within each subsquare. Since the subsquare are much smaller than the whole image, the quantization of the parameters is low and the voting process quicker. The hierarchy is used to group local evidence of short line segments into the more global evidence of larger ones. The grouping is itself a Hough voting process; each segment on a level votes for a possible line orientation in the subsquare of the parent. The higher the level, the finer the quantization, because a subsquare in higher levels maps a larger region of the image in the base. If the grouping at a given level is not sufficiently strong, the process stops. The overall result is spread out in the levels of the pyramid: short segments are detected in the lower levels, the few long segments in the higher ones. The algorithm thus consists of the true multiresolution matching process.

Another way of using the pyramid structure to solve this problem is proposed by Bongiovanni *et al.*[69] It also starts with a "block computation" phase, similar to the previous one, even if executed on disjoint subsquares. But it can also be classified in the larger category of divide-and-conquer algorithms. The combination of the partial results of the subsquares is carried out using the specialized pyramid move operations of Miller and Stout,[10] essentially to concentrate the partial results in the midlevel of the pyramid. The aggregated accu-

mulator is reconstructed in the base by using the ''funnel read'' pyramid operation.

10.4.4.2. Curves

The multiresolution representation of curves is a very important tool in many applications, notably in geographic information systems. It was therefore intensively studied in variable-resolution hierarchical structures, such as quad trees. For an exhaustive review, see Samet.[70, 71] In what follows, we keep the discussion within the pyramid representation, whereby the resolution changes with each level but is uniform within each successive level.

A first type of representation mainly addresses the topology of the digitized curve, as available when the curve is approximated in a tessellated plane. Kropatsch[72] introduces a set of rules to code a curve on the ''dual pyramid'' (see Section 10.2). These rules are a grammar that establishes how a curve fragment can cross a texel in the plane. By using square tessellations rotated by 45° at each other level, Kropatsch builds a hierarchical grammar, which represents the curve at multiple resolution as words in the alphabet of the terminal symbols of the grammar. The resulting representation shrinks the curve at successive levels and can be used to regularize the curves and filter out sets of small discontinuous fragments.[73]

The approximation of curves in multiresolution was addressed on a monodimensional version of the overlapped pyramid by Narayanan and Rosenfeld.[74] The computed approximation consists of a piecewise interpolation built on consecutive groups of four points. Such groups overlap by 50% so that the level of the hierarchy decreases by a factor of 2 at each level. Furthermore, because of the overlapping, each segment can be linked, at the most, to two parent segments in the level above. The linking strategy is unforced: a segment chooses its parent on the basis of the mean-square error (MSE) between the segment approximations. If the MSE is above a chosen threshold, no link is established. After the linking process, the approximations are recomputed at all levels (as in all algorithms that use linking-driven segmentation), using only the subsegments that are currently linked to a parent to obtain the new approximation at each coarser level. Admittedly, the multiresolution representation of the curve is a noncontinuous polygonal approximation of the waveform.

A peculiar type of curve pyramid was introduced by Meer *et al.*[21] It is called the ''chain pyramid'' to highlight the fact that it requires as input a

bidimensional digitized curve that allows a chain-code representation.[75] The requirement for a digitized curve to be chain codable is that any pixel belonging to the curve has at most two neighboring pixels in the usual 8-connected square tessellation of the plane. The chain pyramid consists of a multiresolution doubled-linked data structure which describes the connectivity of the curve in terms of pointers to adjacent pixels. The construction of such a multiscale curve is based on the 2×2 nonoverlapped pyramid. Pixels in the base belonging to the curve are classified as normal (labeled with 1) and siblingless (labeled with 0). A pixel is siblingless if it is the only one belonging to the curve in the 2×2 subsquare of the plane where it is located. A fundamental property of the chain-codable curve is that the quad-pyramid approximation preserves continuity in the coarse level even if the fine level consists of a string of siblingless pixels. This property is very important in reducing the height of the multiscale curve: indeed, a winding curve that crosses the midline of the finest resolution plane at every other pixel provokes the use of all resolution levels. Meer *et al.* eliminate the sequences of siblingless pixels by applying a stochastic election process to halve them (see Section 10.2). The use of the stochastic algorithm, which was shown to converge very rapidly, is required to obtain a decimation which is independent of any knowledge of the location of the curve in the digitized plane. The overall algorithm is shift invariant.

The chain pyramid was used successfully for contour smoothing. The process consists of computing, at each coarser level, the centroids of finer-resolution line segments that are within the ''receptive field'' of a parent. The smoothed contour is the polygonal line of the centroids. The key point in the algorithm is the improved smoothing that results from the stochastic elimination of siblingless pixels, which introduce local irregularities which propagate through all levels of the representation. An enhanced chain pyramid (built with a 4×4 overlapped pyramid) was used to extract *trends* and *significant extrema* of curves.[76] Trends are the sketches of a curve at a coarse representation.

Gaussian blurred pyramids built on noncontiguous sequences of dots, or dot–line image patches, offer a computational model of the perceptual phenomenon that allows humans to perceive such configurations as curves.[77] The surfaces that result from the blurred Gaussian convolution at multiple levels are analyzed for ridges and peaks. A set of connected local maxima is the support of the perceived curve. The rich set of such maxima must be processed in order to prune the multiple paths. The use of multiple-resolution Gaussian blurring guarantees that the number of local maxima decreases when the resolution decreases, this being a property of the Laplacian edge detector (see Section 2.4.3 for details).

10.4.4.3. Shapes

A generalization of the problem of describing curves is that of describing shapes in general. Much interest has been devoted recently to shape; an up-to-date summary is available in Arcelli *et al.*[78]

The most promising approaches seem to be those that consistently apply multiresolution techniques. The rationale for this tendency is that most shapes are too rich in salient features (corners, points of high curvature, parallelism, symmetries) to be economically described at a single resolution. Moreover, as shown in Section 2.2.1, a multiresolution shape representation may be very effective in object recognition. In fact, suitable models and descriptions in multiresolution may be used as guidelines for emulating the focus-of-attention paradigm typical of human vision.

A very effective means of obtaining the description of curvature in a shape on a fine-grained massively parallel pyramidal structure was proposed by Cantoni and Levialdi.[79] The process of heat diffusion was applied to a multiscale version of the shape, which is assumed to be immersed in an adiabatic system. A uniform heat impulse is associated with the shape boundary at all levels. Few iterations of the diffusion process leave a different ''temperature'' on each boundary point. This temperature is directly related to curvature, because high-curvature points offer a narrower area for the heat to flow in.

Multiscale polynomial approximation of curves have an analytic foundation that allows us to compute various geometric properties that are invariant through the resolutions.[80] Corners, straight and curved arcs with signed curvature, and position of inflection points are some of these properties.

An explicit multiresolution definition of symmetry was recently proposed by Zabrodsky *et al.*[81] on the basis of a ''continuous symmetry measure.'' The multiresolution scheme helps in obtaining an initial approximation of axial symmetry. The main application is recognition of faces.

The generalized version of the Hough transform was used on the pyramid for shape recognition.[82, 83] In this case, the model of the shape is described by many voting rules, with each one associated with a given level of representation. The salient features of the shape behave like edge points in the line detection problem and are used to identify subparts of the shape at a given resolution. These subparts become the voting tokens at the next highest level in the pyramid. The pyramid implements a pipelined recognition process by executing the voting processes starting from the base and moving the accumulated evidence upward.

A popular set of features used for shape description and recognition is the

complete moment set. The moment of a continuous function $f(x, y)$ of order n, where $n=p+q$, is defined as[84]

$$M_{p,\,q} = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} x^p y^q f(x,y)\,dx\,dy, \qquad p,\,q=0,1,2,\ \ldots \tag{10.1}$$

When used to represent a segment of an image, $f(x, y)$ is the image function in the segment region and is assumed to be zero outside the region. With respect to the original pixel domain, a small, fixed number of operations are required in the moments domain to perform transformations such as translation, scale change, and rotation. A complete moment set (CMS) of order n contains all moment of order n and lower. It has been shown by Reeves and Rostampour[85] that a CMS is closed with respect to the above transformations in the 2-D plane. A set of standard moments has been defined which normalizes the CMS with reference to shape, size, position, and orientation.

The multiresolution environment is particularly suitable for CMS computation: a window-centered moment of order n on a level k is easily computed on the basis of a simple combination of window-centered moments of order n and lower orders in level k-1:[86]

$$M_{n_k} = \frac{1}{2^n}\sum_{r=0}^{n}\binom{n}{r}\boldsymbol{w} * M_{r_{k-1}} \tag{10.2}$$

The definition is given in one dimension to simplify notation, and $\boldsymbol{w}$ is a moment-generating kernel which combines a moment arm with the standard Gaussian-generating kernel w introduced in Section 2.4.2. Equation (10.2) can be applied recursively: in this fine-to-coarse implementation cumbersome computations within wide windows are performed by iterating combinations of simpler primitives applied to smaller windows.

10.4.5. Image Compression, Coding, and Transmission

The tapering nature of pyramids was recognized long ago as a basis for decreasing the cost of image transmission.[87] Intuitively, the coarse representation of intensity pyramids can be considered a ''compressed'' version of the image.

Obviously, in the field of compression, two main approaches are possible.

Lossless compression requires a complete representation; lossy compression can relax some constraints and use more pragmatic tools. In Chapter 2 we analyzed the properties of many multiresolution representations. The wavelet representation is the basis for a complete and nonredundant coding scheme. As described, it is intrinsically hierarchical and originated many algorithms that couple the pyramid structure with efficient coding schemes. The detail signals of coarser resolution levels are quantized differently with respect to finer ones, both to obtain a compressed signal and to match physiological evidence of the perceptual sensitivity of the human visual system.

Along the same lines is the fundamental work of Burt and Adelson,[88] which is based on the redundant, though complete, Laplacian pyramid. Extremely efficient lossy schemes based on various orthogonal pyramid structures were proposed by Adelson *et al.*[89] Moreover, the very simple Haar transform can lead to significant compression in the special case of robotic scenes.[90] Attempts to base the compression on special pyramid structures are reported by Mayer and Kropatsch.[91, 92] Even though the dual pyramid with 3×3 support has twice as many levels as the 5×5 Laplacian pyramid of Burt, good compression ratios of up to 1:20 were obtained.

10.4.6. Motion Analysis

The analysis of motion is a complex task that involves many levels of processing and is motivated by quite practical problems, such as automatic surveillance and navigation. The need to obtain real-time estimates of object motion in a scene demands effective computation strategies to reduce the enormous amount of data produced by the transducer.

Multiresolution approaches to motion analysis primarily address this issue. By building a properly tuned set of reduced-resolution images for each incoming scene, they carry out the required computation at the most convenient resolution and refine coarse approximations only where necessary. This approach is quite general but becomes the only possible one when the analysis has to take into account ego motion (a moving camera observing a scene with moving objects) and tracking of specific objects. Section 2.1.2 describes the physiological evidence of the behavior of the human vision system with regard to this type of processing. Later we describe an approach to ego motion and tracking that closely resembles the human vision system operation.

In the remainder of this section we review the main results involving multiresolution motion analysis, starting from the low-level phase of optical

flow computation and proceeding upward to object tracking. Before going through the various proposals, we briefly summarize the most relevant definitions and physical properties of motion estimation.

10.4.6.1. Preliminary Definitions

We can define *image flow field* and *optic flow field,* according to Singh,[93] as follows: the image flow field is the bidimensional projection onto the points of the imaging surface of the three-dimensional instantaneous velocities of the corresponding points in the scene, whereas the optic flow field is the ''2D distribution of *apparent velocities* that can be associated with the variation of brightness patterns on the image.'' It is obvious the second one is measured in a sequence of images. Furthermore, the two flows do not coincide in the most general case: if the scene is static and the light source is moving, the image flow is null, while the optic flow is not.

The conditions under which the two flows can be assumed to be equal have been widely investigated (for a detailed analysis, see Singh[93] and Verri and Poggio[94]). Basically, they involve lambertian surfaces moving in a purely translational motion under an illumination that is spatiotemporally uniform. Under these assumptions, it can be shown that the normal components of image flow and optic flow (that is, the components along the image gradient) are roughly equal at those points of the image where the gradient is sufficiently strong. This is a definite source of uncertainty in motion estimation, yet not the only one.

Indeed, since the majority of techniques for optic flow estimation are based on a *local* analysis carried out on image pixels, the *aperture problem* arises. It consists of the fact that it is possible to compute only the component of optic flow normal to the underlying contour, whereas the tangential one cannot be recovered unless the small area used for the computation contains a reach feature, such as a corner or a textured region. To overcome this problem, various types of ''constraints'' have been introduced: they involve either smoothness constraints, which try to enforce a ''good continuation'' on the approximate solution, or neighborhood analysis. It is beyond the scope of this book to discuss them, and the reader is referred to Singh[93] for a bibliography on the subject.

The estimation of optic flow has been traditionally addressed as a ''generalized matching'' between two consecutive images. To derive the local velocities at each point in image $I(t)$, this image is made to correspond to the succes-

sive image $I(t+\Delta t)$ in a pointwise way. The displacements of corresponding point couples are used to derive the local velocities by using the interframe latency Δt. The basic assumption for this analysis is that the velocities in the scene can be recovered with enough precision despite their possibly different magnitudes. With respect to this issue, multiresolution techniques exhibit their best possibilities: rapidly moving objects can be safely tracked at low resolution, since the required precision on the associated velocities is usually much smaller than that of a slowly moving object. These, in turn, can be tracked accurately at high resolution, because the ''displacements'' of corresponding points are rather small and can be identified through coarse-to-fine strategies, which substantially reduce the complexity of the search (see Sections 2.3 and 2.5 and the Section on matching in this chapter).

In the contest of multiresolution, the generalized matching procedure between two consecutive images has been traditionally solved through gradient-based, correlation-based, and spatiotemporal approaches. Only the first two have been explicitly addressed in a multiresolution environment, and are briefly described in the following section. The spatiotemporal approach, pioneered by Adelson and Bergen,[95] is somewhat related to a unifying proposal by Anandan[96] and will be touched upon in that context.

10.4.6.2. Optical Flow through Image Gradient

The image-gradient approach to optic flow estimation is based on the *brightness constancy* assumption.[97] A point P in the scene projects onto the imaging surface at location (x, y) at time t and intensity value $I(x, y, t)$ equal to the projected intensity $I(x+\delta x, y+\delta y, t+\delta t)$ at the corresponding point on the following image. By denoting by u and v the optical flow components dx/dt and dy/dt and by I_x, I_y, and I_t the variations in space and in time of the intensity, the above-mentioned constancy constraints translates to the following equation:

$$\frac{dI}{dt} = I_x u + I_y v + I_t = 0 \tag{10.3}$$

As anticipated in the discussion on the aperture problem, this single equation in u and v leaves the problem underconstrained. Indeed, only the normal component v_n of the optical flow can be computed through the gradient magnitude $G = \sqrt{I_x^2 + I_y^2}$ as

$$v_n = \frac{-I_t}{\sqrt{I_x^2 + I_y^2}} \tag{10.4}$$

Various approaches have been proposed to overcome this problem. The majority of them consist of introducing a "regularization constraint," such as "smoothness of flow-field" according to Horn and Schunck,[97] or "oriented smoothness," according to Nagel[98] (who also takes into account second-order derivatives in formulating the differential equation for the brightness constancy criterion). The determination of optical flow is thus reformulated as the minimization of an appropriate functional giving a solution that is as faithful as possible to the brightness constancy criterion, while obeying the regularization constraint. The usual Euler–Lagrange equations are employed to minimize the functional, and a discrete version of the continuous problem is derived. The solution is then found using a Gauss–Seidel relaxation scheme for the large and sparse linear system of equations.

We are interested in those approaches to this method that embed a multiresolution scheme. As anticipated in Section 10.4.1, the multilevel relaxation scheme is a very general technique and has been applied to optical flow determination as a main case. Experiments are reported by Glazer,[28] Terzopoulos,[32] and Enkelmann.[99] The first two adopt the standard multigrid construction described in Section 10.4.1, relying on interpolation for coarse-to-fine data transfers and on subsampling for fine-to-coarse reductions. Enkelmann added to this approach the preliminary construction of Gaussian pyramids on the two image frames. Variations on the control strategy of the relaxation scheme have been exploited to speed up convergence at the finest level.

It turns out that this approach produces fast good solutions to the optical flow problem when the scene contains a single moving object, while it fails in more complex motion patterns. The difficulty was ascribed by Battiti *et al.*[100] to the conflicting situation of multiple, different motion frequencies that are blurred by the bidirectional information flow of the multigrid relaxation procedure. Good local approximations at a given level can be unduly modified by updates introduced by the upward (downward) propagation of partial solutions at neighboring levels.

A proposed alternative[100] involves an adaptive, error-based multiscale scheme. A local measurement of the maximum expected error is used to control the hierarchical flow of data. In a coarse-to-fine manner, the multigrid relaxation process is executed only at those grid locations where the local error is still above a chosen threshold. The remaining locations, where the rough approximation is already good, are "masked off" along with their siblings.

The following relaxation iterations cannot alter the masked locations, which retain their value. The end effect of this method produces optical flow in a hierarchy of grids. An effective implementation of this solution can exploit the locally refined block structured grids introduced by Gannon.[33]

10.4.6.3. Optical Flow through Image Correlation and Matching

Gradient-based methods rely on the principle of pointwise intensity conservation. Instead, the determination of optical flow through image correlation and higher-level token matching tries to enforce the conservation of *local distribution* of intensity.[93] A pixel in the first image frame is matched to the best corresponding candidate pixel in the second frame on the basis of a measure of similarity computed on an extended neighborhood. The set of couples of matching pixels defines the interframe displacement vector field and the resulting optical flow. Multiresolution helps both in the matching process and in tuning the resolution to the proper velocities in the scene.

A multichannel approach suitable for a pipelined implementation has been proposed by Burt *et al.*[101] After constructing Laplacian pyramids on the two image frames, a set of up to 25 directional correlations is carried out independently at all resolutions. The term *motion channel* identifies one such correlation. Since the magnitude of velocities can vary significantly within a scene, the directional channels will respond differently according to resolution, and multiple ''tuned'' channels will be detected at different resolutions. Linear combinations of channel outputs lead to the actual velocities. Burt introduces criteria to measure the confidence of velocity estimates and to detect those situations that give rise to the aperture problem (lack of local contrast). This approach can be effectively implemented on the ''segmented pipeline'' architecture[102] (see Chapter 6 for more details). It hosts the continuous ''flow-through'' of image data in the stages of the pipeline.

A refined version[103] of the same method enhances the correlation-matching phase as follows. A single level is chosen in the Laplacian pyramids, such that the ''sample distance is only slightly larger than the largest residual motion displacement between frames.'' With this choice, the correlation-matching process requires directional channels with unit displacements. The actual correlation match is executed by first building an integration Gaussian pyramid on the nine resulting channels. By choosing a level in these Gaussian pyramids, the cross-correlation function at each point is assembled as the 3×3 array of values from the nine Gaussian pyramids and the local velocity is obtained as the extremum of the surface that best fits the 3×3 array of correlation values. Consis-

tent use of pipeline architecture is possible in this enhanced method. No bottleneck is introduced in the pipeline, since the combination of channel outputs exploits the same pipelined approach as the first phases of the computation.

Other matching approaches involve a unidirectional coarse-to-fine refinement process. This was applied by Anandan and Weiss[104] along with a match criterion based on a Gaussian-weighted sum-of-squared-difference scheme. A confidence measure is used to sort out the multiple possible solutions to the match, and the measure takes into account local evidence of directional information.

The "elastic matching," a procedure introduced by Bajcsy *et al.*[105] as a general tool in image correspondence, was applied by Dengler[106] to the optical flow problem. The equations that describe the deformation of an elastic membrane under external forces are reinterpreted to yield the displacement field constraints between two successive images. Local cross-correlations of the sign of the Laplacian operator in the two image frames are used in the motion case to derive the analogues of the "external forces." Dengler also addresses the aperture problem and the unisotropy in the expected displacement field that is not captured by the straight formulation as elastic matching (this is to be compared with the observation of Burt that multiple different velocities are best detected by "tuned motion channels"). The second problem is overcome through the use of the zero crossings of the Laplacian: the expected displacement field is segmented into homogeneous regions delimited by such zero crossings, and the solution is subject to this additional regional constraint. This method is indeed quite similar to the variational principle underlying gradient-based approaches, since it produces a system of linear equations to be solved. Dengler sketches a multiresolution approach that builds a three-level Laplacian pyramid and then applies a coarse-to-fine refinement procedure, analogous to the multiresolution relaxation method already described.

Matching has been applied to the motion detection problem at a higher level than the straightforward computation of optical flow. A hierarchy of image features (surfaces, surface edges, vertices, and lines) to implement a top-down matching strategy were proposed by Venkateswar and Chellappa.[107] The idea is to derive motion estimation first at the coarse level of surfaces and then to proceed to finer estimates by using the rest of the hierarchy. The scheme is essentially a graph-matching procedure, guided by various matching constraints (disparity in intensity, area and length, orientation, ordering, as well as other topological constraints). The end effect of the procedure is a set of trajectories of vertices.

Another form of high-level matching is proposed by Grosky and Jain.[108] They consider the image intensity as a surface and proceed to match surface

regions in successive frames through a model-based approach. The multiresolution helps in segmenting regions to be matched (the pyramid linking scheme is used in this case; see Section 10.4.2.1 for details). The candidate regions are fitted to an elliptic paraboloid, and the parameters of the best fit are derived in the pyramid structure during the segmentation process. Once corresponding regions are detected, their analytical surface representation allows easy computation of rotational as well as of translational components of the underlying motion.

A unified approach to optical flow estimation was introduced by Anandan.[96] He reviewed the main hierarchical methods outlined above and showed that gradient-based techniques and his own correlation matching scheme[104] converge into a unique method, provided that the interframe time interval tends to zero and that third- and higher-order spatial derivatives are ignored. Then he extends the hierarchical matching approach by including selective filters tuned to different spatial frequencies, image locations, orientation and single motion direction. Each such filter, which actually extends the notion of motion channel introduced by Burt, is a ''tuned'' motion detector, compatible with all approaches to optical flow estimation, including the spatiotemporal one.

10.4.6.4. Ego Motion and Tracking

The determination of the dense flow field associated with optical flow is only a single step in procedures such as tracking moving objects for surveillance tasks or road-border following for automated navigation. The flow-field analysis is the input to more global estimates of motion, which include the detection of moving surfaces while taking into account the relative motion of the camera with respect to a scene.

A comprehensive proposal for the tracking problem by Burt *et al.*[103] models the dynamic motion analysis carried out with a moving camera for tracking multiple moving objects as a three-stage, coordinated process. The local analysis consists of the determination of optical flow from two successive image frames, using the approach already described. The highest level of analysis is the global description of segmented, coherently moving surfaces. The intermediate-level implements the ''foveal'' analysis. This is carried out on a regional basis and consists of deriving a coherent motion description of a single planar surface within the chosen region. The region of analysis, however, is changed dynamically, possibly from frame to frame, to suit the tracking task. Burt names this sequence of regional analysis a ''focal probe.'' The multiresolution approach is applied in the focal analysis as well, because the dimension of the

region and the sample density within the regions are adjusted to produce a roughly constant processing load.

The key idea in the Burt proposal is the tracking scheme that allows us to quickly detect and segment the surface in the region and which serves to solve the ego-motion problem—the delineation of motion due to the moving camera. Tracking is accomplished in multiresolution. At a coarse level, the local flow vectors are fitted to the preliminary rough coherent motion due to a single surface. The parameters of this estimated surface motion are used to ''warp'' the first image frame with respect to the second, thus effectively reducing the residual displacement. A subsequent local analysis at a higher resolution updates the estimate, and a sequence of such tracking-by-warping steps yields the final motion parameters. This process carries out the analysis of motion together with a segmentation of the region on a single surface against the background. If different coherently moving surfaces are present in the scene, no good fit can be obtained, and the procedure requires splitting and refining the region. Thus, tracking is a single-surface-at-a-time process. Nevertheless, this tracking scheme can be readily applied to determine the ego motion of the camera by adopting a ''majority-tracking'' approach. Indeed, if the region contains a single (or no) object in relative motion with respect to the scene and the camera (provided that the object is small with respect to region of analysis), the majority of the low-level flow-field vectors will be due to the motion of the camera with respect to the still scene. By executing the fitting and tracking procedure against these vectors, the motion of the camera can easily be computed. The single moving object is then delineated by a second tracking step.

The overall motion detection scheme is very well suited to cheap implementation on a mixed hardware platform, with dedicated devices for the low-level phase (see the PVM system and/or the Sarnoff pyramid chip in Chapter 6) and off-the-shelf microprocessors for the foveal probe and tracking steps.

Another multiresolution approach to ego-motion determination is pursued by Hanna.[109] Instead of using the optical flow as an intermediate step, as in Burt's scheme, this approach directly relates the brightness constancy constraint to a global ego-motion constraint. Local brightness derivatives and an ego-motion model are combined to match local planar surface evidence in the image. The procedure starts from a chosen level in the Laplacian pyramid of the first image frame. The current coarse estimate of local surface and ego-motion parameters are used to warp the first image frame to the second one. The residual displacement is used to update through a number of iterations the ego-motion estimates derived from the minimization of the global constraint (which takes into account both the brightness constancy criterion and the ego-motion planar surface model). Then, the current solution is projected onto the next

finer level in the pyramid, and this coarse-to-fine scheme is iterated down to the highest resolution. The algorithm reported here reliably determines ego motion in the absence of independently moving multiple objects in the scene. Since this algorithm uses a global approach, Burt's regional tracking method cannot be easily incorporated into it.

10.4.7. Stereo Vision and Depth

Recovering depth information from a scene is an important step in the 3-D interpretation of the scene itself. Passive methods rely on multiple images, taken at different locations, to infer depth through stereo vision. Active methods use sonar and laser range finders to produce a direct measurement of distances from the active sensor. Alternatively, focusing can be used to substitute the emission of signals with a modification of the geometry of image formation in the camera. Multiresolution offers a viable approach both in passive and in active methods. We summarize a few sample algorithms representative of the application of a multiresolution strategy.

10.4.7.1. Depth from Stereo

Stereopsis in the human visual system and the corresponding computation in computer vision have been modeled through a combined approach: the measure of disparity in space (displacement in retinal position of matching features) along with a proper use of frequency channels (respectively, multiresolution representations) allows us to infer the distance in space of the features detected at the receptors by triangulation. The fundamental work on the subject by Marr and Poggio[110] solves the correspondence problem by using zero crossings of the Laplacian of Gaussian filtered images as features to be matched. Multiresolution comes into the method through the use of multiple Gaussian convolution filters, each with a different standard deviation.

The importance of multiresolution is highlighted explicitly by a result of Clark and Lawrence.[111] In the context of Witkin's *scale-space* transform,[39] they show that computing spatial disparity by matching zero crossings at a single resolution level leads to errors. Such errors are larger for coarser resolutions. Consequently, a coarse-to-fine procedure, which relies on zero crossing, is somewhat diminished in its capacity to focus toward more precise localizations, since it can start with an initial offsetting error. A further contribution of Clark and Lawrence is the foveal scale-space transform, which is a transformation of the Witkin method, leading to the reduction (and in some cases to the elimination) of disparity errors. The real drawback of such a method lies in its cost.

The scale-space transform is a continuous function of the standard deviation of its Gaussian kernel. A reliable computation of this transform involves many more filterings of the input image than a straightforward multiresolution construction, such as the Gaussian or the Laplacian pyramids.

An analysis of the advantages of multiresolution in solving the matching problem through the epipolar constraint was carried out by Cantoni *et al.*[112] In matching points in a pair of images of the same scene, the epipolar constraint helps to reduce the search complexity from $O(N^2)$ to $O(N)$, N being the linear dimension of the image. Moreover, let P be a point in the scene and P_1 its projection in a first image. If a rough estimate of the distance d of P is already available (i.e., $d_{\min} \leq d \leq d_{\max}$), the search in the epipolar line in the second image for the corresponding point P_2 can be narrowed to a segment. However, if no estimate for d is available, or if the matching segment is a substantial part of the whole epipolar line, it is worth applying a multiresolution coarse-to-fine strategy to isolate the tie point P_2.

Cantoni *et al.* built a pyramid (by straight subsampling) on each of the two image frames and estimated the cost of executing the correlation-matching procedure in top-down fashion. The use of a coarse version of the images speeds up the correlation phase. However, the top-down refinement (projecting, etc.) adds an overhead, which is proportional to the number of resolutions used. Since the coarsest level for the first approximate match itself depends on the rough estimate of the displacement, the overall cost of the procedure (as a function of the number of resolution levels) exhibits a minimum, located somewhere between two and four levels. If more levels are used, the gain due to the narrowed search at the coarser levels is canceled by the overhead of the refinements.

10.4.7.2. Depth from Focus

A pyramid algorithm for depth map reconstruction from a single view of a scene was proposed by Darell and Wohn.[113] The basic idea of recovering depth information from focus is to exploit the intuitive notion of sharpness. A ''properly'' focused image appears sharp; that is, it preserves the high-frequency content of the scene. On the contrary, a defocused image is blurred; it can thus be modeled as the convolution of the focused image with a low-pass filter, such as a Gaussian.

If the scene contains a single object, it is possible to analyze the high-frequency decades of the power spectrum of a set of images taken at various focal distances and to derive a function between the spectral components in those frequencies and the focal distance. Such a function shows a single peak

in correspondence with the ''true'' focal distance. This distance is uniquely related to the distance of the object by the usual optic equations.

However, a scene usually contains a number of objects (or prominent features) at different distances so that no single focus can be derived. Still, if the image can be segmented into regions each associated with a single dominant object, it is possible to apply the method to each region separately, thus creating a small depth map.

Darell and Wohn introduce a ''sharpness criterion'' to perform this analysis of the high-frequency band. The criterion is based on the integral of the convolution of the image with a bandpass filter, which enhances the high-frequency components. They adopt the Laplacian pyramid representation, which is a bandpass transformation at multiple resolutions. To compute the integral, they further build a Gaussian pyramid on a single level k of the Laplacian pyramid. If this procedure is repeated many images over, with each taken at a different focusing distance, the resulting Gaussian pyramids generate a map of ''sharpness'' values at multiple resolutions. By choosing a proper integration level in the Gaussian pyramid, each pixel at level k in the Laplacian pyramid has a set of approximations of the power spectrum of the region of the input image that it represents. The maximum value within such a set shows the best focal distance for that region. The resolution of the depth map depends on the level k at which the integration is executed.

This method was implemented on the PVM system described in Chapter 6. The pipelined computation of the Laplacian and Gaussian pyramids can be executed in real time. The bottleneck in the procedure is the time required to adjust the camera lens to the new focus.

10.5. CONCLUSIONS

In this chapter we have analyzed the effectiveness of pyramidal systems in solving computer vision problems. We have considered basic algorithms, such as sorting, connected component labeling, histogramming, and more advanced ones, such as segmentation, object delineation, shape description and recognition, image compression and transmission, motion analysis, and tracking.

This review has highlighted the computational complexity of the algorithms, to show the gain due to the pyramid topology, and their suitability to the different architectural solutions: homogeneous hierarchies versus heterogeneous ones. The class of homogeneous pyramid has received more attention; this is due partly to the fact that compact and pipelined pyramids have been the

first architectures proposed for embedding the multiresolution approach into a parallel systems, partly because they have been conceived as general purpose systems, while heterogeneous ones are more tailored to specific tasks.

The number of algorithms designed for hierarchical systems or embedding a multiresolution approach is ever increasing. The rich set of alternative hierarchical systems offers a wide platform for successfully solving many relevant vision problems; so, it is easy to find a proper matching between the algorithms and the architectures.

REFERENCES

1. S. L. Tanimoto and A. Klinger (eds.), *Structured Computer Vision: Machine Perception through Hierarchical Computation Structures,* Academic Press, New York (1980).
2. A. Rosenfeld, (ed.) *Multiresolution Image Processing and Analysis,* Springer-Verlag, Berlin (1984).
3. V. Cantoni and S. Levialdi (eds.), *Pyramidal Systems for Computer Vision,* Springer-Verlag, Berlin (1986).
4. L. Uhr (ed.), *Parallel Computer Vision,* Academic Press, Orlando, FL (1987).
5. C. R. Dyer, Multiscale image understanding, in *Parallel Computer Vision* (L. Uhr, ed.), pp. 171–213, Academic Press, Orlando, FL (1987).
6. A. Rosenfeld, Image analysis and computer vision: 1991, *CVGIP: Image Understanding* **55**(3), 349–380 (1992).
7. M. Ferretti, Overlapping in compact pyramids, in *Pyramidal Systems for Computer Vision* (V. Cantoni and S. Levialdi, eds.), pp. 247–259, Springer-Verlag, Berlin (1986).
8. R. Miller and Q. F. Stout, *Parallel Algorithms for Regular Architectures,* MIT Press, Cambridge, MA (1992).
9. Q. F. Stout, Pyramid algorithms optimal for the worst case, in *Parallel Computer Vision* (L. Uhr, ed.), pp. 147–168, Academic Press, New York (1987).
10. R. Miller and Q. F. Stout, Data movement techniques for the pyramid computer, *SIAM Comput.* **16** (1), 38–60 (1987).
11. S. L. Tanimoto, Algorithms for median filtering of images on a pyramid machine, in *Computing Structures for Image Processing* (M. J. B. Duff, ed.), pp. 123–141, Academic Press, London (1983).
12. S. L. Tanimoto, Sorting, histogramming, and other statistical operations on a pyramid machine, in *Multiresolution Image Processing and Analysis* (A. Rosenfeld, ed.), pp. 136–145, Springer-Verlag, Berlin (1984).
13. Q. F. Stout, Sorting, merging, selecting and filtering on tree and pyramid machines, *Proc. 1983 Int. Conf. Parallel Processing,* 1983, pp. 214–221.
14. Q. F. Stout, Supporting divide-and-conquer algorithms for image processing, *J. Parallel Distribut. Comput.* **4,** 147–168 (1987).
15. S. L. Tanimoto, Programming techniques for hierarchical parallel image processors, in *Multicomputers and Image Processing Algorithms and Programs* (K. Preston and L. Uhr, eds.), pp. 421–429, Academic Press, New York (1982).
16. R. Miller and Q. F. Stout, Computing convexity properties of images on a pyramid computer, *Algorithmica* **6,** 659–684 (1991).

17. R. Miller and Q. F. Stout, Simulating essential pyramids, *IEEE Trans. Comput.* **TC-37**(12), 1642–1648 (1988).
18. W. G. Kropatsch, Rezeptive felder in bildpyramiden, in *Mustererkennung 1988* (H. Bunke, O. Kübler, and P. Stucki, eds.), pp. 333–339, Springer-Verlag, Berlin (1988).
19. W. G. Kropatsch, A pyramid that grows by powers of 2, *Pattern Recognition Lett.* **3**(9), 315–322 (1985).
20. P. Meer, Stochastic image pyramids, *CVGIP* **45,** pp. 269–294 (1989).
21. P. Meer, C. A. Sher, and A. Rosenfeld, The chain pyramid: hierarchical contour processing, *IEEE Trans. Pattern Anal. Machine Intell.* **PAMI-12**(4) 363–376 (1990).
22. P. Meer, S. Jiang, E. S. Baugher, and A. Rosenfeld, Robustness of image pyramids under structural perturbations, *CVGIP* **44,** 307–331 (1988).
23. A. Montanvert, P. Meer, and A. Rosenfeld, Hierarchical image analysis using irregular tessellations, *IEEE Trans. Pattern Anal. Machine Intell.* **PAMI-13**(4) 307–316 (1991).
24. J. M. Jolion and A. Montanvert, The adaptive pyramid: a framework for 2D image analysis, *CVGIP: Image Understanding* **55**(3), 339–348 (1992).
25. S. Peleg, O. Federbush, and R. Hummel, Custom-made pyramids, in *Parallel Computer Vision* (L. Uhr, ed.), pp. 125–147, Academic Press, New York (1987).
26. Ph. Clermont, Méthodes de programmasion de machine parallèle pyramidale. applications en segmentation d'images, Thèse de Doctorat, Université Paris VII (1990).
27. Ph. Clermont and A. Merigot, Efficient parallel pyramidal primitives for image analysis, in *Progress in Image Analysis and Processing II* (V. Cantoni, M. Ferretti, S. Levialdi, R. Negrini, and R. Stefanelli, eds.), pp. 544–550, World Scientific, Singapore (1992).
28. F. Glazer, Multilevel relaxation in low-level computer vision, in *Multiresolution Image Processing and Analysis* (A. Rosenfeld, ed.), pp. 312–330, Springer-Verlag, Berlin (1984).
29. W. Hackbusch, *Multigrid Methods and Applications,* Springer-Verlag, New York (1985).
30. A. Brandt, Multi-level adaptive solutions to boundary-value problems, *Math. Comp.* **31,** 333–390 (1977).
31. D. Terzopoulos, Multilevel computational processes for visual surface reconstruction, *CVGIP* **24,** 52–96 (1983).
32. D. Terzopoulos, Image analysis using multigrid relaxation methods, *IEEE Trans. Pattern Anal. Machine Intell.* **PAMI-8**(2), 129–139 (1986).
33. D. Gannon, On the structure of parallelism in a highly concurrent PDE solver, *Proc. 7th Symp. Computer Arithmetics,* 1985, pp. 252–259.
34. P. Burt, T. H. Hong, and A. Rosenfeld, Segmentation and estimation of image region properties through cooperative hierarchical computation, *IEEE Trans. Syst., Man, Cybernet.* **SMC-11,** 802–804 (1981).
35. T. Hong, K. A. Narayanan, S. Peleg, A. Rosenfeld, and T. Silberberg, Image smoothing and segmentation by multiresolution pixel linking: further experiments and extensions, *IEEE Trans. Syst., Man, Cybernet.* **SMC-12**(5), 611–622 (1982).
36. J. M. Cibulskis and C. R. Dyer, An analysis of node linking in overlapped pyramids, *IEEE Trans. Syst., Man, Cybernet.* **SMC-14**(3), 424–436 (1984).
37. T. Hong and A. Rosenfeld, Compact region extraction using weighted pixel linking in a pyramid, *IEEE Trans. Pattern Anal. Machine Intell.* **PAMI-6**(2), 222–229 (1984).
38. M. Spann, Figure/ground separation using stochastic pyramid relinking, *Pattern Recognition* **24**(10), 993–1002 (1991).
39. A. P. Witkin, Scale-space filtering, Proc. 7th Int. Joint Conf. Artificial Intelligence, 1983, pp. 1019–1021.

40. A. L. Yuille and T. A. Poggio, Scaling theorems for zero crossings, *IEEE Trans. Pattern Anal. Machine Intell.* **PAMI-8**(2), 15–25 (1986).
41. T. Hong, M. Shneier, and A. Rosenfeld, Border extraction using linked edge pyramids, *IEEE Trans. Syst., Man, Cybernet.* **SMC-12**(5), 660–668 (1982).
42. T. H. Hong and M. Shneier, Extracting compact objects using linked pyramids, *IEEE Trans. Pattern Anal. Machine Intell.* **PAMI-6**(2), 229–237 (1984).
43. R. Hartley, A Gaussian-weighted multiresolution edge detector, *CVGIP* **30,** 70–83 (1985).
44. P. Burt, Fast filter transforms for image processing, *CVGIP* **16,** 20–51 (1981).
45. R. Park and P. Meer, Edge-preserving artifact-free smoothing with image pyramids, *Pattern Recognition Lett.* **12**(9), 467–475 (1991).
46. A. Rosenfeld and A. Sher, Detection and delineation of compact objects using intensity pyramids, *Pattern Recognition* 21, 147–151 (1988).
47. C. A. Sher and A. Rosenfeld, Pyramid cluster detection and delineation by consensus, *Pattern Recognition Lett.* **12**(9), 477–482 (1991).
48. P. Meer, D. Mintz, A. Montanvert, and A. Rosenfeld, Consensus vision, Proc. AAAI-90 Workshop on Qualitative Vision, Boston, MA, 1990, pp. 111–115.
49. J. M. Jolion, P. Meer, and A. Rosenfeld, Border delineation in image pyramids by concurrent tree growing, *Pattern Recognition Lett.* **11**(2), 107–115 (1990).
50. L. Van Gool, P. Dewaele, and A. Oosterlinck, Texture analysis anno 1983, *CVGIP* **29,** 336–357 (1985).
51. M. Pietikäinen and A. Rosenfeld, Image segmentation by texture using pyramid node linking, *IEEE Trans. Syst., Man, Cybernet.* **SMC-11**(12), 822–825 (1981).
52. L. I. Larkin and P. Burt, Multi-resolution texture energy measures, *Proc. IEEE Comput. Soc. Conf. CVPR,* Washington, DC, 1983, pp. 519–520.
53. B. P. Kjell and C. R. Dyer, Edge separation and orientation texture measures, Proc. IEEE Conf. CVPR, 1985, pp. 306–311.
54. B. P. Kjell and C. R. Dyer, Segmentation of textured images by pyramid linking, in *Pyramidal Systems for Computer Vision* (V. Cantoni and S. Levialdi, eds.), pp. 273–288, Springer-Verlag, Berlin (1986).
55. S. Peleg, J. Naor, R. Hartley, and D. Avnir, Multiple resolution texture analysis and classification, *IEEE Trans. Pattern Anal. Machine Intell.* **PAMI-6**(4), 518–523 (1984).
56. D. B. Mandelbrot, *The Fractal Geometry of Nature,* Freeman, San Francisco, CA (1982).
57. C. Bouman and B. Liu, Multiple resolution segmentation of textured images, *IEEE Trans. Pattern Anal. Machine Intell.* **PAMI-13**(2), 99–113 (1991).
58. M. Bister, J. Cornelis, and A. Rosenfeld, A critical view of pyramid segmentation algorithms, *Pattern Recognition Lett.,* **11**(9), 605–617 (1990).
59. A. Rosenfeld and G. J. VanderBrug, Coarse-fine template matching, *IEEE Trans. Syst., Man, Cybernet.* **SMC-7**(2), 104–107 (1977).
60. R. Y. Wong and E. L. Hall, Sequential hierarchical scene matching, *IEEE Trans. Comput.* **C-27**(4), 359–366 (1978).
61. S. L. Tanimoto, Template matching in pyramids, *CVGIP* **16,** 356–369 (1981).
62. F. Glazer, G. Reynolds, and A. Anandan, Scene matching by hierarchical correlation, Proc. IEEE CS Conf. CVPR, Washington, DC, 1983, pp. 432–441.
63. P. V. C. Hough, Method and means for recognizing complex patterns, U.S. Patent 3069654 (1962).
64. D. Ballard, Generalizing the hough transform to detect arbitrary shapes, *Pattern Recognition* **13**(2), 111–122 (1981).

65 H Li, M A Lavin, and R J Le Master, Fast hough transform a hierarchical approach, *CVGIP* **36,** 139–161 (1986)
66 S L Tanimoto, From pixels to predicates in pyramid machines, in *From Pixels to Features* (J C Simon, ed), pp 383–392, Elsevier, North-Holland (1989)
67 J M Jolion and A Rosenfeld, A $O(\log n)$ pyramid Hough transform, TR-2066, Center for Automation Research, University of Maryland, College Park, MD (1988)
68 J Princen, J Illingworth, and J Kittler, A hierarchical approach to line extraction based on the hough transform, *CVGIP* **52,** 57–77 (1990)
69 G Bongiovanni, C Guerra, and S Levialdi, Computing the Hough transform on a pyramid architecture, *Machine Vision Appl* **3**(2), 117–123 (1990)
70 H Samet, *The Design and Analysis of Spatial Data Structures,* Addison-Wesley, Reading, MA (1990)
71 H Samet, *Applications of Spatial Data Structures Computer Graphics, Image Processing, and GIS,* Addison-Wesley, Reading, MA (1990)
72 W G Kropatsch, Curve representations in multiple resolution, *Pattern Recognition Lett* **6**(8), 179–184 (1987)
73 W G Kropatsch, Elimination von 'kleinen' kurvenstucken in der $2 \times 2/2$ kurvenpyramide algorithmus und test, DIBAG-Report Nr 25, Institut fur Digitale Bildverarbeitung und Grafik, Graz (1987)
74 K A Narayanan and A Rosenfeld, Approximation of waveform and contours by one-dimensional pyramid linking, *Pattern Recognition* **15**(5), 389–396 (1982)
75 H Freeman, Computer processing of line-drawing images, *Comput Surveys* **6,** 57–97 (1974)
76 P Meer, E S Baugher, and A Rosenfeld, Extraction of trend lines and extrema from multiscale curves, *Pattern Recognition* **21**(3), 217–226 (1988)
77 S Connelly and A Rosenfeld, A pyramid algorithm for fast curve extraction, Center for Automation Research Tech Report CAR-TR-270, University of Maryland (1987)
78 C Arcelli, L P Cordella, and G Sanniti di Baja (eds), *Visual Form Analysis and Recognition,* Plenum Press, New York (1992)
79 V Cantoni and S Levialdi, Contour labeling by pyramidal processing, in *Intermediate-Level Image Processing* (M J B Duff, ed), pp 181–190, Academic Press, London (1986)
80 A Bengtsson and J Eklundh, Shape representation by multiscale contour approximation, *IEEE Trans Pattern Anal Machine Intell* **PAMI-13**(1), 85–93 (1991)
81 H Zabrodsky, S Peleg, and A Avnir, Hierarchical symmetry, Proc 11th Int Conf Pattern Recognition, Vol C, 1992, pp 9–12
82 L S Davis, Hierarchical generalized Hough transform and line-segment based Hough transform, Technical Report, University of Texas (1979)
83 V Cantoni, L Carrioli, M Diani, M Ferretti, L Lombardi, and M Savini, Object recognition and location by a bottom-up approach in *Image Analysis and Processing* (V Cantoni, V Di Gesù, and S Levialdi, eds), pp 329–336, Plenum Press, New York (1988)
84 M-K Wu, Visual pattern recognition by moment invariants, *IIRE Trans Inform Theory* **IT8,** 179–187 (1962)
85 A P Reeves and A Rostampour, Shape analysis of segmented objects using moments, Conf Pattern Recognition and Image Processing, Dallas, 1981, pp 171–174
86 P J Burt, Smart sensing within a pyramid vision machine, *Proc IEEE* **76**(8), 1006–1015 (1988)
87 K R Sloan and S L Tanimoto, Progressive refinement of raster images, *IEEE Trans Comput* **C-28**(11), 871–874 (1979)

88. P. J. Burt and E. H. Adelson, The Laplacian pyramid as a compact image code, *IEEE Trans. Commun.* **COM-31**(4), 532–540 (1983).
89. E. H. Adelson, E. Simoncelli, and R. Hingorani, Orthogonal pyramid transforms for image coding, SPIE, Vol. 845, Visual Communications and Image Processing II, 1987, pp. 50–58.
90. M. G. Albanesi, I. De Lotto, and L. Carrioli, Image compression by the wavelet decomposition, *European Trans. Telecommunications,* **3**(2), 45–54 (1992).
91. H. Mayer and W. G. Kropatsch, Progressive bildübertragung mit der $3 \times 3/2$ pyramide, in *Informatik Fachberichte 219: Mustererkennung 1989* (H. Burkardt, K. H. Köhne, and B. Neumann, eds.), pp. 160–167, Springer-Verlag, Hamburg (1989).
92. H. Mayer and W. G. Kropatsch, Kompakte bildkodierung mit der $3 \times 3/2$ pyramide, in *Wissenbasierte Mustererkennung* (A. Pinz, ed.), pp. 195–210, Oldenbourg, Austria (1989).
93. A. Singh, *Optic Flow Computation,* IEEE Computer Society Press, Los Alamitos, CA (1991).
94. A. Verri and T. Poggio, Against quantitative optical flow, Proc. First ICCV, London, 1987, pp. 171–180.
95. E. H. Adelson and J. R. Bergen, Spatio-temporal energy models for the perception of motion, *J. Opt. Soc. Am. A* **2**(2), 284–299 (1985).
96. P. Anandan, A unified perspective on computational techniques for the measurement of visual motion, Proc. 1st ICCV, 1987, pp. 219–230.
97. B. K. P. Horn and B. Schunck, Determining optical flow, *Artif. Intell.* **17,** 185–203 (1981).
98. H. H. Nagel, Displacement vectors derived from second order intensity variations in image sequences, *CVGIP* **21,** 85–117 (1983).
99. W. Enkelmann, Investigation of multigrid algorithms for the estimation of optical flow fields in image sequences, *CVGIP* **43,** 150–177 (1988).
100. R. Battiti, E. Amaldi, and C. Koch, Computing optical flow across multiple scales: an adaptive coarse-to-fine strategy, *Int. J. Comput. Vision* **6**(2), 133–145 (1991).
101. P. J. Burt, C. Yen, and X. Xu, Multi-resolution flow-through motion analysis, Proc. IEEE CS Conf. CVPR, Washington, DC, 1983, pp. 246–252.
102. P. J. Burt, Multiresolution pyramid architectures for real-time motion analysis, IAPR Workshop on Machine Vision Applications, Tokyo, 1990, pp. 317–321.
103. P. J. Burt, J. R. Berger, R. Hingorani, R. Kolczynski, W. A. Lee, A. Leung, J. Lubin, and H. Shvaytser, Object tracking with a moving camera, Proc. IEEE Workshop on Visual Motion, Princeton, NJ, 1991, pp. 2–12.
104. P. Anandan and R. Weiss, Introducing a smoothness constraint in a matching approach for the computation of displacement fields, Proc. SPIE Intelligent Robots and Computer Vision Conf., **521,** 1984, pp. 184–194.
105. R. Bajcsy, R. Lieberson, and M. Reivic, A computerized system for the elastic matching of deformed radiographic images to idealized atlas images *J. Comp. Assoc. Tomography* **7**(4), 618–625 (1983).
106. J. Dengler, Local motion estimation with the dynamic pyramid, in *Pyramidal Systems for Computer Vision* (V. Cantoni and S. Levialdi, eds.), pp. 289–298, Springer-Verlag, Berlin (1986).
107. V. Venkateswar and R. Chellappa, Hierarchical feature based matching for motion correspondence, Proc. IEEE Workshop on Visual Motion, Princeton, NJ, 1991, pp. 280–285.
108. W. I. Grosky and R. Jain, Region matching in pyramids for dynamic scene analysis, in *Multiresolution Image Processing and Analysis* (A. Rosenfeld, ed.), pp. 331–342, Springer-Verlag, Berlin (1984).

109. K. J. Hanna, Direct multi-resolution estimation of ego-motion and structure from motion, Proc. IEEE Workshop on Visual Motion, Princeton, NJ, 1991, pp. 156–162.
110. D. Marr and T. Poggio, A computational theory of human stereo vision, *Proc. R. Soc. London B* **204,** 1979, pp. 359–365.
111. J. J. Clark and P. D. Lawrence, A theoretical basis for diffrequency stereo, *CVGIP* **35,** 1–19 (1986).
112. V. Cantoni, A. Griffini, and L. Lombardi, Stereo vision in multi-resolution, in *Progress in Image Analysis and Processing* (V. Cantoni, L. P. Cordella, S. Levialdi, and G. Sanniti di Baja, eds.), pp. 706–713, World Scientific, Singapore (1990).
113. T. Darell and K. Wohn, Depth from focus using a pyramid architecture, *Pattern Recognition Lett.* **11**(12), 787–796 (1990).

Index