Pyramidal Systems for Computer Vision

NATO ASI Series

Advanced Science Institutes Series

A series presenting the results of activities sponsored by the NATO Science Committee, which aims at the dissemination of advanced scientific and technological knowledge, with a view to strengthening links between scientific communities.

The Series is published by an international board of publishers in conjunction with the NATO Scientific Affairs Division

A Life Sciences B Physics	Plenum Publishing Corporation London and New York
C Mathematical and Physical Sciences	D. Reidel Publishing Company Dordrecht, Boston, Lancaster and Tokyo
D Behavioural and Social Sciences E Applied Sciences	Martinus Nijhoff Publishers Boston, The Hague, Dordrecht and Lancaster
F Computer and Systems Sciences G Ecological Sciences H Cell Biology	Springer-Verlag Berlin Heidelberg New York London Paris Tokyo



Series F: Computer and Systems Sciences Vol. 25

Pyramidal Systems for Computer Vision

Edited by

Virginio Cantoni

Pavia University, Istituto di Informatica e Sistematica
Strada Nuova 106/C, 27100 Pavia/Italy

Stefano Levialdi

Rome University, Dipartimento di Matematica,
Piazzale Aldo Moro 2, 00185 Rome/Italy



Springer-Verlag
Berlin Heidelberg New York London Paris Tokyo
Published in cooperation with NATO Scientific Affairs Divison

Proceedings of the NATO Advanced Research Workshop on Pyramidal Systems for Image Processing and Computer Vision at Maratea, Italy, May 5–9, 1986

ISBN-13:978-3-642-82942-0 e-ISBN-13:978-3-642-82940-6
DOI: 10.1007/978-3-642-82940-6

Library of Congress Cataloging in Publication Data. Pyramidal systems for computer vision. (NATO ASI series. Series F, Computer and systems sciences ; vol. 25) Proceedings of the NATO Advanced Research Workshop on Pyramidal Systems for Computer Vision, held May 5–9, 1986, in Maratea, Italy. 1. Computer vision—Congresses. I. Cantoni, V. II. Levialdi, S. III. NATO Advanced Research Workshop on Pyramidal Systems for Computer Vision (1986 : Maratea, Italy) IV. North Atlantic Treaty Organization. Scientific Affairs Division. V. Series: NATO ASI series. Series F, Computer and systems sciences ; v. 25. TA1632.P97 1986 006.3'7 86-27904
ISBN-13:978-3-642-82942-0 (U.S.)

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically those of translating, reprinting, re-use of illustrations, broadcastings, reproduction by photocopying machine or similar means, and storage in data banks. Under § 54 of the German Copyright Law where copies are made for other than private use, a fee is payable to "Verwertungsgesellschaft Wort", Munich.

© Springer-Verlag Berlin Heidelberg 1986
Softcover reprint of the hardcover 1st edition 1986

2145/3140-543210

Preface

This book contains the proceedings of the NATO Advanced Research Workshop held in Maratea (Italy), May 5–9, 1986 on *Pyramidal Systems for Image Processing and Computer Vision*.

We had 40 participants from 11 countries playing an active part in the workshop and all the leaders of groups that have produced a prototype pyramid machine or a design for such a machine were present.

Within the wide field of parallel architectures for image processing a new area was recently born and is growing healthily: the area of pyramidally structured multiprocessing systems. Essentially, the processors are arranged in planes (from a base to an apex) each one of which is generally a reduced (usually by a power of two) version of the plane underneath: these processors are horizontally interconnected (within a plane) and vertically connected with "fathers" (on top planes) and "children" on the plane below. This arrangement has a number of interesting features, all of which were amply discussed in our Workshop including the cellular array and hypercube versions of pyramids.

A number of projects (in different parts of the world) are reported as well as some interesting applications in computer vision, tactile systems and numerical calculations.

The design of pyramidal algorithms, based on primitive operations, on the upward and downward propagation of the information (from base to apex and vice versa) and the possibility of executing digital transforms with pyramids was investigated and may be found in these proceedings as well as image segmentation (based on a model of the visual system), local motion estimation, graphical applications and the detection of vectorial features.

A review of existing programming language paradigms for image processing and a specific pyramidal language are presented to account for the software side of the computation.

In closing, the VLSI technology is analyzed with reference to the manufacturing of multiprocessor pyramid architectures and the availability of different strategies for performing fault-tolerance on such complex systems is discussed.

The friendly atmosphere of the workshop and the synergism obtained in the different informal meetings that we held at Maratea has given us a strong encouragement to pursue these research lines until these machines reach the production line.

Virginio Cantoni and Stefano Levialdi
Rome, 1986

Table of Contents

Parallel, Hierarchical Software/Hardware Pyramid Architectures
Leonard Uhr 1

I.P. Hierarchical Systems: Architectural Features
Virginio Cantoni 21

General Purpose Pyramidal Architectures
Gerhard Fritsch 41

Pyramids – Expected Performance
Michael J.B. Duff 59

Hypercubes and Pyramids
Quentin F. Stout 75

Architectural Comparisons
Serge Castan 91

A Pyramidal System for Image Processing
A. Merigot, P. Clermont, J. Mehat, F. Devos and B. Zavidovique 109

Counting on the GAM Pyramid
David H. Schaefer and Ping Ho 125

A Pipelined Pyramid Machine
P.J. Burt, C.H. Anderson, J.O. Sinniger and G. van der Wal 133

The PAPIA Controller: Hardware Implementation
Gaetano Gerardi 153

Custom Made Pyramids
Shmuel Peleg and Orna Federbush 165

Paradigms for Pyramid Machine Algorithms
Steven L. Tanimoto 173

Pyramid Algorithms on Processor Arrays
Anthony P. Reeves 195

Pyramidal Transforms in Image Processing and Computer Vision
Philipp W. Besslich 215

Overlapping in Compact Pyramids
Marco Ferretti 247

Some Pyramid Techniques for Image Segmentation
Azriel Rosenfeld 261

Segmentation of Textured Images by Pyramid Linking
Bradley P. Kjell and Charles R. Dyer 273

Local Motion Estimation with the Dynamic Pyramid
Joachim Dengler 289

Vectorial Features in Pyramidal Image Processing
Hans G. Zimmer 299

Programming Image Processing Machines
Stefano Levialdi 311

A High Level Language for Pyramidal Architectures
Vito Di Gesù 329

Tactile Information Processing
Ruzena Bajcsy and Sharon Stansfield 341

Silicon Implementation of Multiprocessor Pyramid Architecture
Franco Maloberti 357

Fault-Tolerance Techniques in Arrays for Image Processing
Roberto Negrini and Renato Stefanelli 373

PARALLEL, HIERARCHICAL SOFTWARE/HARDWARE PYRAMID ARCHITECTURES

Leonard Uhr
Department of Computer Sciences
University of Wisconsin
Madison, Wisconsin 53706, USA

Introduction

This paper examines pyramid-structured software/hardware systems, both programs and multi-computers. It explores how efficient parallel computers can be designed. It poses the extremely difficult problem of perception, and briefly surveys the major alternative approaches. It examines the basic structures with which pyramids can be built, and describes pyramid multi-computers. It sketches out how pyramid hardware can be used to execute programs, with special emphasis on the "recognition cone" structures being developed to model living visual systems. It briefly suggests how pyramids can be augmented and embedded into larger software/hardware structures. In addition, it gives a short history of massively parallel hierarchical pyramid structures.

Serial and Parallel Computers, General-Purpose and Specialized

The classical serial 1-CPU Von Neumann computer is capable of scanning over any possible program graph (simply because any graph can be decomposed into a set of 1-node graphs). But it is guaranteed to take the longest possible time.

The only way the single-CPU Von Neumann computer that has for the past 40 years been such a tremendous success can be substantially further improved in power and in speed is to combine large numbers of individual computers into parallel networks. This is because the VLSI technologies that are about to pack increasing millions of transistors onto each tiny silicon chip are fast approaching the speed of light.

A great variety of different types of multi-computers are being investigated, designed, and built. But it is not at all clear whether it will be possible to design and build multi-computer networks that will execute any and all programs with high efficiency and sufficient speed. It may well be necessary to settle for reasonable trade-offs between power, generality, efficiency, speed, and cost-effectiveness.

NATO ASI Series, Vol. F 25
Pyramidal Systems for Computer Vision
Edited by V. Cantoni and S. Levialdi
© Springer-Verlag Berlin Heidelberg 1986

True Data-Flow Hardware That Is Isomorphic to the Program's Structure

Possibly the ideal is to map the program into an isomorphic data-flow hardware. This would be equivalent to building the appropriate finite state automaton that was equivalent to that program. Brains appear to be systems of this sort, since they are networks of neurons through which data flow. However, it is not at all clear that such a system could be generally usable over a wide range of different programs.

There are a number of other interesting possibilities. They are just beginning to be investigated, and much work will be needed to develop any of them.

Systems that Flow Data Through a Smaller Sub-Graph

Rather than build hardware whose structure is isomorphic to the entire program graph, a much smaller hardware graph can be constructed. Now that graph can be used to scan over the larger program graph. This is exactly what is today done with the single-CPU computer. Since it is a 1-node graph any program graph can be decomposed into it, for successive processing. (Note that this gives the slowest possible execution, since the single processor can do only one thing at a time, serially.)

In order to increase speed, the number of nodes in the hardware graph must be increased substantially. It is not known whether this is possible in the general case, although there is one striking commercial success - the linear pipeline of processors that give today's "supercomputers" their great power at the vector processing so common in scientific computing.

Systems that Use a General Topology that Can Execute ALL Programs Efficiently

The program can be represented as a graph, and the multi-computer can be represented as a graph. The problem becomes one of finding a multi-computer graph into which all program graphs can be mapped with reasonable efficiency. Since it is not known what different graph topologies are needed for different programs of interest, it is impossible to judge whether such a system can be realized. It may well be that there are a small number of basic structures that programmers could use to compose large programs.

Systems that Use Switches to Reconfigure Among Appropriate Topologies

Switches might be incorporated into the hardware network, so that it could be reconfigured to handle each different program input to it. A generally reconfigurable system is in theory possible. But the cost in switches would probably be too great. (These costs increase as the network grows larger, and will quickly dominate the cost of the entire system.)

Networks that Contain Suitably Specialized Regions

Several different structures can be combined together, to build the larger network. Reconfiguring switches can be used to link the individual structures together in a variety of different ways, as directed by the program.

Some General Issues

Different programs will, inevitably, be of different sizes and with different amounts of parallelness. Therefore the only way to use a general-purpose multi-computer efficiently is to multi-program, so that different programs will use otherwise idle resources. This further complicates the problem.

However, it may well be that percent utilization is unimportant. Transistors and chips get cheaper and cheaper. Increasingly important will be speed and power.

This paper concentrates on program-structured specialized systems, along with attempts to make them more general. The perception program motivates and suggests the hardware.

The Perception Problem Posed, and its Extreme Difficulty

The perceptual recognition and description of complex real-world objects is an extremely difficult problem. Probably half of the many billions of neurons in brains of higher mammals are involved with perception. The speed at which perception takes place is incredibly fast.

The Massively Parallel and Shallowly Serial Perceptual System

The human visual system has about 10,000,000 cones and 100,000,000 rods in each eye. The cone system is sensitive to color and shape. The rod system is sensitive to change, whether of intensities in the spatial domain or of position in the temporal domain. Cones are concentrated in the central fovea, where shapes are perceived. Rods extend out to the periphery, and thus serve as an early warning system.

The living perceptual system is capable of recognizing and describing complex scenes of complex objects in from 30 to 800 milliseconds. But the basic cycle time of a neuron - that is, the amount of time it takes to bridge the synapses over which neurons fire other neurons - is roughly 1 or 2 milliseconds. This means that the serial depth of the perceptual system is only a few hundred steps at most, and possibly only a few dozen. The visual system appears to be organized in layers of neurons that converge and diverge information, as they transform that information, moving inward from retina to primary and then to a number of secondary

visual areas, and then other parts of the cortex (see Uhr, 1980, 1986c for more extended examinations of these issues).

The System Must Recognize and Describe Scenes of Complex Moving Objects

To recognize, describe, and track real-world objects as they move about and interact with one another in real time, the perceptual system must be able to process each input (e.g., each frame in a succession of TV images) in only 30 milliseconds or so. This means that the serial depth when processing moving images is especially shallow.

The System Must Handle Enormous Amounts of Information in Complex Scenes

Real-world scenes typically contain a number of different objects. Psychological experiments indicate that we human beings are able to recognize objects and their distinguishing and salient characteristics when there are dozens in the single scene. This is a largely nonconscious process that takes place in a few hundred milliseconds.

The System Must Recognize Highly Structured Objects Over Unknown Variations

Each object is complex, with a large number of individual details, some of which are of importance and must be noticed. Each instance of each object can vary in a wide variety of ways. The linear transformations (translation, magnification, rotation) are only the simplest. Far more important, and more difficult to handle, are the unknown non-linear transformations effected by atmospheric distortions, muscles that twist, emotions, and aging. Consider the many ways that a face can change as it smiles, frowns, or looks tired or discouraged.

The Major Alternative Software Approaches

A large number of programs have been written to recognize objects (variously called image processing, pattern recognition, and computer vision). Most of these fall into one or another of the following general types.

A) Programs that Combine Results of Independent Procedures (Feature Detectors)

In the simplest case, the image is processed with a number of procedures (often called feature-detectors), and their results combined into a decision as to what might be there.

A highly parallel 2-dimensional array structure is often used to realize such systems.

When executed on hardware arrays, these systems typically use a variety of simple, usually local, features. (Non-local features can take extremely long and inefficient sequences of operations.)

These systems are usually considered to be bottom-up, first applying feature detectors, then combining their results, and finally choosing among alternative possibilities. But they can also apply feature-detectors in a top-down manner, or be given a top-down component.

They can be extremely fast. But they are relatively weak, and suited only for the recognition of small, simple objects that vary little.

B) Programs that Assess Syntactic-Like Structures Over Primitives

A number of programs have been developed that attempt to use 2-dimensional extensions of grammars, giving tree-like structures. However, there are major unsolved problems when trying to generalize grammars that have been developed for linear strings to handle 2-dimensional objects. In a linear string only one substring can be linked (concatenated) to the previous substring. That is, each substring touches only two other substrings, one on either side. In sharp contrast, in a 2-dimensional space a thing (feature or object) might join to a potentially infinite number of other things.

Such syntactic recognition systems usually expect and can handle only images made up of perfectly connected simple primitive curves. This means that they work only on carefully pre-processed or toy data (e.g., line drawings of simple objects).

C) Programs that Match Stored Models with Appropriate Regions in the Image

Probably the most commonly proposed type of system for "high-level" computer vision is one that attempts to match models of each possible object in a top-down manner. Typically, this kind of system serially matches models (commonly called "frames," or "schemata") to sub-regions of the image. The model is, essentially, a graph - and, for real-world objects, this will be an extremely complex graph. The image must be processed, usually in a bottom-up manner, until it is converted into a similar graph. This usually means that its edges, angles, contours, and regions must all be identified and correctly linked to one another.

This is extremely difficult (if not impossible - consider the prefectly recognizable images that can be constructed by randomly erasing most of the regions and edges). In any case, the system is, essentially, faced with a graph-matching problem. (Note that in the general case this is an NP-complete problem, since the graph that models the object must be matched with a sub-graph of the graph into which the image has been transformed.)

This kind of system is typically given only a few models, and matches them separately, one by one. It is usually considered to be top-down. Such systems can handle only a few ob-

jects, and are inevitably slow.

D) Programs that Hierarchically Examine Successively Larger Structures

Highly parallel, logarithmically serial "pyramid/cone" structures appear to offer real hope of overcoming the problems of other types of vision systems.

They can make use of local operations that are executed in parallel everywhere. This means that they can be quite general-purpose and robust, looking for and being able to respond to, unanticipated objects. They can combine bottom-up and top-down processing, moving upward, downward, and laterally through the pyramid. Therefore they can be looking for the most expected objects, while at the same time monitoring the environment for the unexpected. And they can keep gathering information and testing tentative hypotheses until enough information has been accumulated and assessed to come to a decision.

They can combine and apply many models, that are all embedded in the pyramid, in a relatively efficient way. If each model is turned into a tree-like graph, with the proper number of levels, these graphs can be embedded into a pyramid so that each hardware level executes about the same number of instructions. That is, the model can be decomposed into a hierarchy of procedures that are embedded into hardware to load-balance across levels. Now all the different models of different objects can be squashed together. Wherever they have nodes (that is, processes) in common these can all be combined, so that the process need be executed only once. This will often be the case, especially at the lower levels, where the same gradient, edge, texture, and region detectors will be common to and appropriate for recognizing many different objects. They therefore appear to be capable of handling many objects, potentially with speeds great enough to handle the real-time perception of moving objects.

Developing Programs and Architectures for Fast, Efficient Execution

A large variety of different network topologies have been developed for multi-computers. An enormously larger variety is possible, since the network's underlying topology can be any possible graph.

The program graph must flow smoothly through the computer structure. That is, the flow of processes that makes up the program must be decomposable into the graph or graphs that make up the multi-computer that will execute those processes.

This means that for perception, the program must examine large 2-D images, on the order of 512 by 512, 1,024 by 1,024, or even larger. For scenes of moving objects, each image must be processed in only 20 to 50 milliseconds, or even less.

In the general case the perceptual system can't anticipate what might be there, or where. It

must look everywhere, and therefore must have a massively parallel bottom-up component to its processing.

The following begins to describe the several major types of multi-computer architectures that have been developed. It emphasizes structures that are most appropriate for perception, presenting them in a natural progression that leads up to an examination of pyramids. The section below on augmenting pyramids will briefly explore Then we will examine additional architectures with which pyramids can fruitfully be combined.

The Conventional 1-CPU Serial Computer (SISD)

The classic single-CPU serial computer is, essentially, a 1-node graph.

Any program graph can be flowed through it, but serially. Except for very small and simple problems, or toy demonstrations, this is much too slow for perception.

1-Dimensional Strings: Pipes (MISD), Rings, Arrays (Synchronized SIMD)

Several individual computers can be linked together to form a string. That is, each computer links to two other computers (except for the first and the last, which each link to only one). Systems of this sort have been built with as many as 1,024 processors.

Possibly the simplest way to use such a system is as a linear pipeline through which information is streamed. Each computer in the pipe repeatedly executes the same process, but on different sets of information - much as each worker in an assemblyline repeatedly executes his assigned task. Two major types of specialized pipelines have been designed, built, and successfully commercialized.

The first type is the number cruncher "super-computers" like the Crays and Cybers (see Riganati and Schneck, 1984). They are given pipelines of 10 or so very powerful 64-bit floating point processors to operate on vectors of numbers.

The second type has been specialized for image processing. These include PICAP (Kruse, 1976) and the Cytocomputers (Sternberg, 1978; Lougheed and McCubbrey, 1985), and also a number of commercial pipelined scanning arrays like the deAnza and Vicom. Each has 1 to 1,024 8-bit processors, specialized to do 8-bit fixed-point arithmetic on grey-scale and color images. Additional hardware allows for very fast scanning through the 2-dimensional array that contains an image - whether the raw image, or a transformed representation.

When a P-node pipe is kept full and busy, the system can get up to P-fold speed-ups, minus the time needed to fill and to empty the pipe.

The last node of a pipeline of this sort can be linked back to the first, forming a ring. The system can now cycle and pipe the last processor's results back around into the first, and this cycling can continue potentially without limit.

Alternately, the 1-dimensional line can be used to process a 2-dimensional array of data, by piping each row (or column) of the 2-dimensional array through the appropriate different processor in the line. For example, the PIXIE-5000's 1 by 1,024 array of 1-bit processors can process a 1,024 by 1,024 array in 1,024 instructions (taking 80 microseconds in toto).

2-Dimensional Arrays (Most are SIMD, But They Needn't Be)

The multi-computers that have actually been built with the largest number of processors (by far) are the 2-dimensional SIMD arrays. These are especially appropriate for perception, where the basic input is the very large 2-dimensional image.

The 2-dimensional array can directly process 2-dimensional data. It can also process larger arrays, and N-dimensional arrays.

It is important to mention a much smaller 2-dimensional array of each far more powerful 64-bit floating point processors - the number cruncher ILLIAC-IV (Barnes et al., 1968). This was the largest and most powerful computer built during the early 1970s.

A number of 2-dimensional arrays with a much larger number of processors have been built more recently. These include the 96 by 96 CLIP4 (Duff, 1976), the 64 by 64 DAP (Reddaway, 1978), and the 128 by 128 MPP (Batcher, 1980). Each computer links directly to its 4 or 8 nearest neighbors. All execute the instruction broadcast by a single controller, and therefore run in SIMD mode. All use relatively simple (but general-purpose) 1-bit processors, with 4 to 8 processors on each chip.

This kind of array can be given more and more computers, potentially without limit. For example, a 1,024 by 1,024 TV image can be input to a 1,024 by 1,024 array of computers, so that each contains one cell of the image. Now each processor can compute functions of local regions of information surrounding the spot stored in its memory. Successive functions will therefore compute successively more abstract results about this local region. Several processors can be used, if desired, to process each region. For example, each of 8 different oriented edges could be looked for at the same time, if 8 processors were assigned to each spot.

An array of processors smaller than the image array can iterate serially through the larger array. For example, a 1,024 by 1,024 array of computers can process a 4,096 by 4,096 image array by storing a 4 by 4 subarray in each computer, and serially iterating through the 16 spots. Alternately, each of the 16 1,024 subarrays can be processed in turn.

Arrays have very good local connectivity. When they are given a single controller this is nicely combined with very good global coordination, so that a massively large number of local operations can all be executed at the same time. For example, the program can in one step look for a local feature like a gradient, edge, color, or texture everywhere, in many thousands or millions of places. When several controllers are used, one for each subarray or reassignable under program control, the array can execute several different processes, as appropriate,

in different regions of the scene.

Arrays can become quite slow when they must make global assessments, or pass information over great distances. This is so because the only way that an array can compute a function of information stored at several different locations is to shift all the pieces of information to a single processor. To do this a serial sequence of shifts must be effected. In the worst case this can mean, in an N by N array with direct links from each processor to its 4 square neighbors (to North, South, East and West), up to 2N shifts for each piece of information.

Pyramid of Arrays (SIMD-MIMD)

A pyramid can most simply be described as a stack of successively smaller arrays that are linked together by a tree. For example, a 16 by 16 array might have each 2 by 2 subarray linked to the corresponding cell of an 8 by 8 array of parents. These similarly link to a 4 by 4 array, and this linking scheme continues to a 2 by 2 and a 1 by 1 (apex) array.

Pyramids can be designed in a large number of different ways (see Uhr, 1984, 1986c). For example, pyramids with square bases can converge 2 by 2, 3 by 3, or N by N, or N by M then M by N. They can have rectangular or hexagonal bases, and converge appropriately. They can converge different amounts at different levels. They can have overlap, with each node linked to several parents, or no overlap, with each node linking to only one parent. Since information can be passed laterally through the array links at each layer, overlap can be simulated quite easily by a non-overlapped pyramid. Similarly, an overlapped pyramid does not need links to siblings, which can be reached by moving down to children, and then back up.

Each array in the pyramid can execute all array operations efficiently. In addition, passing information up and down between arrays needs greatly reduced logarithmic distances, rather than the linear distances needed within an array. For example, whereas a 1,024 by 1,024 array needs up to 2,046 shifts to send information, when augmented with a pyramid (which adds less than 1/3d more computers) it needs at most 20. This is a crucial difference in terms of the extremely small amount of time available for real-time perception of real-world objects.

The major purpose of a pyramid is not to communicate, but to compute. Rather than simply pass information, or average and compress information, moving up the pyramid, the processors can be programmed to process and transform that information in any way desired.

Using Pyramid Software/Hardware Systems

Pyramid-like structures can be built and used in several different ways.

Possibly the simplest is the quad-tree, which is a tree superimposed over an array of infor-

mation. (Note that this is not a pyramid, since it has neither lateral links to siblings or overlap that allows for communication via children.) Wherever a parent node's children all contain the same value that value can be passed up to the parent and the children eliminated. The resulting quad-tree will (to the extent that images do not have many high frequency variations) be much reduced in size, yet it can be used to reconstruct the image exactly.

Possibly the simplest way to use a true pyramid is to successively reduce the image by averaging. This can serve for data compression. It also makes possible a quick global assessment at a higher level, so that the program can then zoom down and look in more detail at particular regions that it judges may be of interest. Or the program can apply feature-detectors of the appropriate frequency to each of the different levels.

Rather than average the raw image, the program can apply a feature detector, e.g., for gradients, edges, or textures, and then successively average their output.

A number of relatively efficient pyramid algorithms have been developed for such operations as region growing, blob counting, median filtering, and contour and region linking (Cibulski and Dyer, 1984; Miller, 1984; Stout, 1983, 1985; Tanimoto, 1983). These make use of the array links when appropriate, and of the up and down tree links to span larger distances. It is important to emphasize that the converging tree structure moving toward the pyramid's apex, which on the one hand gives the pyramid its good logarithmic information-passing distances, can cause severe bottlenecks when too much information must be passed. For example, pyramids, because of their tree structure, are poor for sorting and permuting information.

Using Pyramid Multi-Computers to Execute "Recognition Cone" Processes

The approach that I and my associates have been taking to using pyramids (Uhr, 1971, 1973, 1975; Uhr and Douglass, 1979; Schmitt, 1981; Li and Uhr, 1985a, 1985b, 1986) attempts to model what living perceptual systems appear to do as they successively process and converge information, leading toward recognition. The overall pyramid structure is an extraction from and simplification of the much larger system, which also has parallel arrays, diverges information, and fans out into the semantic memory and associative areas.

The massively parallel and shallowly serial pyramid structure offers many possibilities for parallelizing and speeding up perceptual processes, and for decomposing the extremely complex procedures needed to perceive real-world objects into a hierarchical structure of simple processes. The basic process is that of a probabilistic threshold element similar to a neuron that fires when enough neurons fire into it.

The model of each object to be recognized is decomposed and converted into a hierarchical tree of suitably simple processes, and then embedded into the pyramid. Each of these processes is made as simple and as local as possible. The many nodes that different object-

models have in common (particularly at the lower levels) are combined into one, so that all objects can be looked for in parallel.

The system can be primed in advance to look for particular objects, or (when processing a sequence of images of a scene of moving objects) it can use what information it has uncovered to help direct its search. Thus a top-down component can be combined with the bottom-up processing that is driven by the image input to the pyramid's base.

For example, structures of transforms of the following sort have been embedded into a 512 by 512-based 2x2 converging 9-layered pyramid.

1) Smooth; Eliminate Specks of Noise.
2) Get Gradients.
3) Get Short Edges, Colors, and Simple Textures.
4) Get Longer Edges, Simple Curves, More Complex Textures, Small Regions.
5) Get Angles, More Complex Curves, Larger Regions.
6) Get Simple Enclosures, and Simple Objects.
7) Get More Complex Objects.
8) Get Still More Complex Objects, and Groups of Objects.
9) Get Still More Complex Objects, and Groups of Objects.

Such systems have successfully recognized real-world objects like neurons, trees, windows, doors, roofs and houses.

A complete program may entail several phases of processing that move up, down, and around the pyramid. The system can be programmed to look downward for objects that are already stored in a "Lookfor" list, as well as do bottom-up processing of a new image. (No matter how strong is the top-down component, a general vision system must always be alert for new, unanticipated but possibly important, objects.) This bottom-up processing will begin to imply new objects that might be present. This in turn starts up new top-down processes to gather the additional information needed to recognize those objects. The system can keep cycling in this way until enough objects have been recognized, or no time is left.

When the system must process a continuing stream of images of objects in motion, there may well be little time for anything but bottom-up processing (remember, to model living visual systems there is time for only a few dozen, or a few hundred, instructions). Therefore it seems best to augment the basic pyramid with additional hardware, to further parallelize and to simultaneously execute bottom-up and top-down processes.

The Need to Augment the Pyramid

There are several reasons why it seems desirable to augment a pyramid, either by adding internal hardware or by embedding it in a larger network. SIMD pyramids (this is also true

for arrays) become inefficient when all processors are not working fruitfully. The tree bottlenecks cannot always be avoided by commensurate reductions of information. Different types of processors may be indicated for different purposes (e.g., to match masks, combine weights, choose among alternatives). It might be desirable to combine several pyramids, for example to handle different sensory modalities. Finally, the perceptual system must be combined with the larger cognitive system.

There are a number of interesting possible augmentations (see Pfeiffer, 1985; Tanimoto, 1985; Uhr, 1983, 1985a, 1985b, 1986a, 1986b). For example, processors can be made reconfigurable to, e.g., 1 64-bit (possibly organized to handle in parallel an 8 by 8 window operation), 4 16-bit, 8 8-bit, or 64 1-bit processors. Several processors can be placed at each cell. Several pyramids can be linked together. The pyramid can, wherever appropriate, be given more communication bandwidth, processor power and/or memory size. Processors can be reassignable under program control to top-down and/or bottom-up flow.

An MIMD network can be linked to the pyramid, at its base, apex, higher layers near the apex, or both high and low layers (now the MIMD network can serve for communication and feedback at the same time that it transforms and processes information). There are a large number of potentially attractive MIMD networks that might be used (see Uhr, 1984, 1986c).

Several networks of this sort can be built to give the programmer the impression that all processors share a common memory. The simplest link all computers via a common bus, ring or crossbar switch. But more than a few dozen computers linked in this way will overload the bus or ring, or need an excessively expensive crossbar. An NlogN switching network can be used to link larger numbers of computers - up to a few hundred, or possibly (but with rapidly increasing costs for the switching network) even a few thousand.

Point-to-point linked networks can be built in a great variety of topologies, since any connected graph can be used. Among the most popular today are N-dimensional hypercubes, trees, and several bus-based or crossbar-based clusters linked together. Such systems have the major advantage that they can, potentially, be built to any size. But there can be difficulties in passing information through a path of intervening processors.

There are a number of other potentially more attractive topologies that have not yet been used. There are a variety of compounding operations that appear to be better than N-cubes (which are compounds of N-1-cubes). A tree that is carefully augmented, usually at each leaf (which in a tree has only one link, to its parent), can be given substantially more computers within a given diameter (the shortest longest distance between nodes) and degree (the maximum number of links to each node). Good constructions of this sort include De Bruijn graphs and Hypertrees. There are also a variety of small graphs with good properties, including the optimally dense Moore graphs, that can be used individually or compounded into larger structures. There are almost certainly many other good constructions that have not even been discovered.

A Short History of Massively Parallel Hierarchical Structures (Pyramids)

The following, as requested by the editors of this volume, briefly sketches a bit of the history of pyramids, including data-structures, structures of software processes (algorithms and programs), and hardware architectures.

The first attempts to develop pyramid-like systems were in the form of software programs and algorithms. Much more recently, people have begun to design and build appropriate pyramid hardware.

Programs that Simulate Pyramid Structures

Kelly (1971) described a system that reduced (by averaging) an image up to a much smaller array, and then used this to determine which parts of the image to process in detail. It thus took a top-down approach to processing.

Klinger (1971, 1974) developed techniques for using quad-tree structures to represent images. Klinger and Dyer (1976) extended systems of this sort and examined how well they worked in actual experimental situations.

Uhr (1971, 1972) described hierarchically converging layered "recognition cone" systems that looked everywhere in a large array for successively more global structures of information. The transforming procedures used were motivated by the threshold element-like neuron, and had much the form of IF...THEN... production rules augmented with weights, thresholds, and 2-dimensional structural relations. An example of such a system was programmed in SNOBOL and given enough transforms to recognize a few simple objects. Later SNOBOL programs (Uhr, 1973, 1976) demonstrated how cone/pyramid structures could be used to describe as well as recognize objects in static scenes, and to begin to handle objects as they move about in real time.

Hanson and Riseman (1974, 1976, 1978) developed "preprocessing cones" that are used primarily for lower levels of semantically directed vision systems.

Tanimoto (1976, 1978) developed pyramids as data structures and, potentially, hardware structures. He was probably the first person to design a hardware pyramid (see below), and a language for describing and programming pyramid structures (1983a, 1984). He has also worked extensively on pyramid algorithms (e.g., 1983b).

Bajcsy and Rosenthal (1975, 1980) examined how to focus attention using hierarchical systems. Sloan (1977), working with Bajcsy, developed a multi-level hierarchical system that applied radically different types of processes at each level.

Levine and Leemet (1976) and Levine (1978, 1980) developed a pyramid program to recognize real-world objects using several different pyramids, each containing appropriate simple features (e.g., color).

Kruse (1976, 1980) used multi-resolution techniques within a pyramid structure to program his PICAP pipeline image processor to zoom in to regions of special interest. This approach was taken in programs to recognize finger prints and to examine circuit boards.

At about the time that Dyer went to Maryland, the group of vision researchers headed by Rosenfeld began to study pyramids (see, e.g., Rosenfeld and Vanderbrug, 1977; Davis and Rosenfeld, 1978, Rosenfeld, 1983). A large number of research papers from Maryland have explored pyramids, chiefly examining multi-resolution (Burt, 1984), quad-trees (Dyer et al., 1980; Samet, 1980), and linking of regions or feature elements (Burt et al., 1981; Pietikainen and Rosenfeld, 1981; Cibulski and Dyer, 1984).

Dyer (1979) also began to investigate pyramids from a theoretical point of view, and more recently to develop pyramid algorithms and programs (e.g., Neveu, Dyer and Chin, 1985). More recently, Stout (1983, 1985) and Miller (1984) have greatly extended this kind of theoretical analysis.

Uhr (1978), Uhr and Douglass (1979), Schmitt (1981), and Li and Uhr (1985a, 1985b, 1986) developed programs to recognize real-world images of houses, neurons, and other objects. These programs begin to extend cone/pyramid systems to use successively more global structures at higher semantic levels, and to combine bottom-up and top-down processing.

Several recent books (Tanimoto and Klinger, 1980; Rosenfeld, 1984; Uhr, 1987) have collected research on pyramids.

The essence of pyramid perception systems lies in the massively parallel (hence extremely fast) application of a converging hierarchy of micro-modular (hence relatively local) operations. Several of the most interesting and the most powerful computer vision systems that are not ordinarily thought of as using pyramids have many of these properties, for example, Tenenbaum and Barrow (1976), Marr (1982), Ballard (1985).

The Design and Construction of Pyramid Multi-Computers

To achieve the great speed that massively parallel hierarchical converging structures potentially offer, specialized pyramid hardware architectures are highly desirable, if not necessary.

Hardware designs for pyramids were first developed by Dyer (1981, 1982), Tanimoto (1981) and Uhr (1981, 1984). Several hardware pyramids are presently being built.

Tanimoto (1983a) is starting to build a large pyramid with each processor linked to 8 siblings, 4 children, and 1 parent. He has also simulated this system, and has designed a language that allows the programmer to work closely with the pyramid structure.

Cantoni et al. (1985) are working with a group of seven Italian Universities and several industrial firms. They have designed and are fabricating a chip that contains 1 parent and 4 children, and are investigating how to make the system fault tolerant.

Schaefer (1985), using MPP chips (each with 8 processors and a fast shift register), has completed a pyramid with a 16 by 16 base.

Merigot et al. (1985) have designed and are beginning to build a pyramid with the topology of a binary tree. This allows convergence at each level to one half (rather than one quarter) the previous level's processors.

All these systems use custom VLSI designs with several processors on each chip. All use 1-bit processors of the sort used by the CLIP, DAP and MPP arrays. All use a single controller for each layer (some may use a single controller for all layers).

Handler and his associates (Handler, 1975; Bode et al., 1985) have been building much smaller pyramids of much more powerful independent computers. Each computer has its own controller, so that the system runs in MIMD mode. These systems have been designed to handle numerical problems. But they should be quite useful for image processing and pattern recognition. Potentially, an MIMD pyramid can offer greater flexibility in applying different processes at different regions of the image, and allocating more resources where appropriate.

Closely Related Hardware Architectures of Interest

Shaw's (1982, 1986) original Non-Von computer, which links 8-bit processors via a tree, has recently been augmented so that the bottom ply (the leaves of the tree) also link laterally to siblings. This gives some of the pyramid properties but, because there is neither overlap nor lateral linkage at higher levels, it is probably closer to a quad-tree, and therefore in danger of having crucial information lost in or separated by cracks.

An interesting and ambitious new 4-level system has been started at the University of Massachussetts (Levitan, personal communication). It will have a very large array of 1-bit processors at its base, 16-bit processors at the next level each linked to an 8 by 8 array of children, and 32-bit Lisp processors at the third level, each linked to a number of children. At the top level, all the Lisp processors will link to a single host.

Several other types of parallel computer appear to be appropriate for executing pyramid programs with appreciable increases in speed over the serial computer. They will not be as fast as true pyramid hardware. But for many purposes they may be more cost effective, or more flexible.

The PIPE pipelined image processor (Kent et al., 1985) can be programmed to remap data into successively smaller arrays, and therefore to execute converging pyramid procedures (see Kent and Tanimoto, 1985). Its 8 stages allow the programmer to build and process a pyramid with a 256 by 256 base. The PIPE can also be programmed to execute up to 256 different instructions at different locations in the array. This is accomplished by storing instructions in one of the array's two 8-bit memories.

The new Cyto-HSS (Lougheed and McCubbrey, 1985) similarly allows the programmer to

address and remap data and therefore build and process pyramids. It has the added advantage that this can be done at the rate of 10 million 3-by-3 window operations per second, rather than the 20 or 30 millisecond TV scan rate for the entire image. As the arrays grow smaller moving up the pyramid this speedup becomes extremely significant.

The PIXIE-5000 (Wilson, 1985) can also remap data in the direction of its scan in one instruction, and in the other direction using a short sequence of instructions. Since it takes only 80 microseconds to process a 1,024 by 1,024 array, in contrast to the 30 milliseconds or so needed by a pipelined scanning array like the Cyto and PIPE, it will still be extremely fast. It will be possible to use CLIP7, which is now being built and will scan a 1-dimensional array of 8-bit processors over the 2-dimensional image array, in a similar manner (Fountain, 1983, 1985).

An array like CLIP4 (Duff, 1976), DAP (Reddaway, 1978), or the MPP (Batcher, 1980) can be programmed to handle pyramids by either squeezing the higher layers into one corner, tiling the higher layers over the array, or scattering the elements of the higher layer over the array. If the bottom several layers of the pyramid being simulated are larger than the hardware array, and subarrays are stored in each processor's memory, the distances information must be shifted laterally through the array in order to simulate a pyramid will be reduced, and processing speeded up significantly (see Reeves, this volume).

An N-dimensional hypercube of the sort being sold by INTEL and NCUBE can be used in the same way (see Stout, this volume), since an array can conveniently be mapped into it. Such a system has the additional advantage of the N-cube linkage, which will cut down the distances that information must be passed. It will also have the MIMD system's added flexibility. On the other hand, MIMD systems are expensive, and the time taken to send information in a message-passing system of this sort is typically thousands of times longer than the time needed for a simple lateral shift in an array or a pyramid. Additional hardware (as planned for the NCUBE company's new systems) should reduce this to a few hundred times longer - a much shorter but still costly amount of time.

Uhr et al. (1983) developed a detailed design for, but never built, a 2-layer system where processors could be reconfigured to either an array of 32-bit MIMD computers or an MIMD-SIMD array of 8-bit computers. Since memory was shared, these could conveniently be used for pyramid processes. Sandon (1985) designed an array that can be reconfigured to have successively more powerful processors moving up through what can be used as a pyramid structure.

Other attractive MIMD systems link all processors via an NlogN reconfiguring network - e.g., PASM (Siegel, 1981) and the Butterfly (Crowther et al., 1985). These can be used even more conveniently than the hypercubes, since any computer can pass information to any other over the switching network that links them together. However, the costs of the switches grow rapidly as the number of computers in the network grows.

Since these MIMD systems use far more expensive processors and communication channels than would a massively parallel pyramid they would inevitably be smaller, hence slower. For example, given the packing densities that will almost certainly be achieved in VLSI technologies during the next 5 to 10 years - of 10 million or more transistors per chip - 256 or 1,024 1-bit processors can be put on each chip. But at least 8 chips would be needed for a conventional serial computer (almost all for memory). Therefore whereas it may be feasible to fabricate a 1,024 by 1,024 array for the base of a pyramid using only 1,024 chips, only a few hundred or a few thousand conventional computers could be fabricated with a few thousand chips and linked together into an N-cube, reconfigurable network, or some other topology. This suggests that the fastest and most cost effective architecture will be one that combines large pyramids of synchronized computers with a suitably structured network of more powerful and more independent processors.

Summary Comments

This paper examines the development of appropriately process-structured software/hardware systems for the perceptual recognition of real-world objects in real time.

The perception program can be represented as a data-flow graph, and the problem of developing an appropriate multi-computer topology treated as one of finding a structure of computers that can efficiently execute that graph as it scans over it. A 1-node graph (the 1-CPU computer) can scan over any graph; but it is far too slow. A 1-D pipeline (e.g., PICAP, Cyto, PIPE) can effectively execute local operations on arrays of iterated information of any dimension, reducing the time needed by the pipe's length. A 2-D array (e.g., CLIP, DAP, MPP) can handle 2, 3, and N-D arrays with speed-ups proportional to the number of processors - but they can be very slow at moving information together for global operations. A pyramid of arrays has all the capabilities of an array for massively parallel local processes, and also the good global properties of tree-based logarithmic structures.

Pyramids can be used with great potential power and efficiency, by treating them as 2-D pipelines through which information is flowed and successively transformed. Several different types of complex real-world objects (e.g., neurons, trees, windows, houses) have been successfully recognized in this way.

A pyramid of arrays becomes inefficient when different processes must be applied in different regions, and when different sub-sets of information must be flowed in different directions. There are a number of possible augmentations - both internal to the pyramid and in the form of additional networks of computers into which the pyramid can be embedded - that offer promise of further substantial increases in power and in generality.

References

[1] R.K. Bajcsy and D.A. Rosenthal, "Visual focussing and defocussing - an essential part of the pattern recognition process," *Proc. IEEE Conf. on Computer Graphics, Pattern Recognition and Data Structures*, 1975.

[2] ----, "Visual and conceptual focus of attention" in *Structured Computer Vision: Machine Perception Through Hierarchical Computation Structures*, S. Tanimoto and A. Klinger, Eds. New York: Academic Press, 1980, 133-149.

[3] D.H. Ballard, "Task frames in visuo-motor coordination," *Proc. Third Workshop on Computer Vision*, IEEE Computer Society Press, 1985, 3-10.

[4] G.H. Barnes, R.M. Brown, M. Kato, D.J. Kuck, D.L. Slotnick and R.A. Stokes, "The ILLIAC IV Computer," *IEEE Trans. Computers*, 1968, *17*, 746-757.

[5] K.E. Batcher, "Design of a massively parallel processor," *IEEE Trans. Computers*, 1980, *29*, 836-840.

[6] A. Bode, G. Fritsch, W. Handler, W. Henning, F. Hofmann and J. Volkert, "Multi-grid oriented computer architecture," *Proc. Int. Conf. Parallel Proc.*, 1985, 89-95.

[7] P.J. Burt, "The pyramid as a structure of efficient computation" in *Multiresolution Image Processing and Analysis*, A. Rosenfeld, Ed. New York: Springer-Verlag, 1984, 6-35.

[8] P.J. Burt, T.H. Hong, and A. Rosenfeld, "Segmentation and estimation of image region properties through cooperative hierarchical computation," *IEEE Trans. System, Man, Cybernetics*, 1981, *SMC-11*, 802-809.

[9] M. Pietikainen and A. Rosenfeld, "Image segmentation by texture using pyramid node linking," *IEEE Trans. System, Man, Cybernetics*, 1981, *SMC-11*, 822-825.

[10] V. Cantoni, S. Ferretti, S. Levialdi and F. Maloberti, "A pyramid project using integrated technology" in *Integrated Technology for Image Processing*, S. Levialdi, Ed. London: Academic Press, 1985, 121-133.

[11] J.M. Cibulski and C.R. Dyer, "An analysis of node linking in overlapped pyramids," *IEEE. Trans. Syst., Man, Cybernetics*, 1984, *14*, 424-436.

[12] W. Crowther, J. Goodhue, E. Starr, R. Thomas, W. Milliken and T. Blackadar, "Performance measurements on a 128-node Butterfly parallel processor," *Proc. Int. Conf. Parallel Proc.*, 1985, 531-540.

[13] L.S. Davis and A. Rosenfeld, "Hierarchical relaxation" in *Computer Vision Systems*, A.R. Hanson and E.M. Riseman, Eds. New York: Academic Press, 1978, 101-109.

[14] M.J.B. Duff, "CLIP4: a large scale integrated circuit array parallel processor," *Proc. 4th Int. Joint Conf. on Pattern Recognition*, 1976, *4*, 728-733.

[15] C.R. Dyer, "Augmented cellular automata for image analysis," *Unpubl. Ph.D. Diss.*, Dept. of Computer Science, Univ. of Maryland, 1979.

[16] -----, "A quadtree machine for parallel image processing," *Information Engin. Dept. Tech. Rept. KSL 51*, Univ. of Illinois at Chicago Circle, 1981.

[17] -----, "Pyramid algorithms and machines" in *Multicomputers and Image Processing*, K. Preston and L. Uhr, Eds. New York: Academic Press, 1982, 409-420.

[18] C.R. Dyer, A. Rosenfeld and H. Samet, "Region representation: boundary codes from quadtrees," *Comm. ACM*, 1980, *23*, 171-179.

[19] T.J. Fountain, "The development of the CLIP7 image processing system," *Pattern Recognition Letters*, 1983, *1*, 331-339.

[20] -----, "Plans for the CLIP7 chip" in *Integrated Technology for Image Processing*, S. Levialdi, Ed. London: Academic Press, 1985, 199-214.

[21] W. Handler, "A Unified Associative and Von-Neumann Processor - EGPP and EGPP Array," *Lectures Notes in Computer Sci., vol. 24 - Parallel Processing, 97-99*, Springer-Verlag, 1975.

[22] M.D. Kelly, "Edge detection in pictures by computers using planning" in *Machine Intelligence 6*, R. Meltzer and D. Michie, Eds. New York: Elsevier, 1971, 379-409.

[23] E.W. Kent, M. Shneier and R. Lumia, "PIPE - Pipelined image processing engine," *J. Parallel and Distributed Computing*, 1985, *2*, 50-78.

[24] E.W. Kent and S.L. Tanimoto, "Hierarchical cellular logic and the PIPE processor: structural and functional correspondence," *Proc. Workshop on Computer Architecture for Pattern Analysis and Image Database Management*, IEEE Computer Society Press, 1985, 311-319.

[25] B. Kruse. "The PICAP picture processing laboratory," *Proc. 4th Int. Joint Conf. on Pattern Recognition*, 1976, *4*, 875-881.

[26] -----, "System architecture for image analysis" in *Structured Computer Vision: Machine Perception Through Hierarchical Computation Structures*, S. Tanimoto and A. Klinger, Eds. New York: Academic Press, 1980, 169-212.

[27] M.D. Levine, "A knowledge-based computer vision system" in *Computer Vision Systems*, A. Hanson and E.

Riseman, Eds. New York: Academic Press, 1978, 335-352.
[28] -----, "Region analysis with a pyramid data structure" in *Structured Computer Vision*, S. L. Tanimoto and A. Klinger, Eds. New York: Academic, 1980, pp. 57-100.
[29] M.D. Levine and J. Leemet, "A method for nonpurposive picture segmentation," *Proc. 4th Int. Joint Conf. on Pattern Recognition*, 1976, *4*, 494-498.
[30] Z.N. Li and L. Uhr, "Comparative Timings for a Neuron Recognition Program on Serial and Pyramid Computers," *Proc. Workshop on Computer Architecture for Pattern Analysis and Image Data Base Management*, IEEE Computer Society Press, 1985, 99-106. (a)
[31] -----, "Pyramidal Algorithms for Analysis of House Images," *Proc. Intersoc. Conference on Artificial Intelligence Applications*, 1985. (b)
[32] -----, "A pyramidal approach for the recognition of neurons using key features," *Pattern Recognition*, 1986, *19*, 55-62.
[33] R.M. Lougheed and D.L. McCubbrey, "Multi-processor architectures for machine vision and image analysis," *Proc. Int. Conf. Parallel Proc.*, 1985, 493-497.
[34] D. Marr, *Vision*, San Francisco: Freeman, 1982.
[35] A. Merigot, B. Zavidovique, and F. Devos, "SPHINX, a pyramidal approach to parallel image processing," *Proc. Workshop on Computer Architecture for Pattern Analysis and Image Database Management*, IEEE Computer Society Press, 1985, 107-111.
[36] R. Miller, Pyramid Computer Algorithms, *Unpubl. Ph.D. Diss.*, Dept. of Math., SUNY Binghamton, 1984.
[37] C.F. Neveu, C.R. Dyer and R.T. Chin, "Object recognition using Hough pyramids," *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, 1985, 328-333.
[38] J.J. Pfeiffer, Jr., "Integrating low level and high level computer vision," *Proc. Workshop on Computer Architecture for Pattern Analysis and Image Database Management*, IEEE Computer Society Press, 1985, 119-125.
[39] S.F. Reddaway, "DAP - a flexible number cruncher," *Proc. 1978 LASL Workshop on Vector and Parallel Processors*, Los Alamos, 1978, 233-234.
[40] J.P. Riganati and P.B. Schneck, "Supercomputing," *Computer*, 1984, *17*, 97-113.
[41] A. Rosenfeld, "Pyramids: multiresolution image analysis," *Proc. Third Scandinavian Conference on Image Analysis*, July, 1983, 23-28.
[42] A. Rosenfeld, Ed., *Multiresolution Image Processing and Analysis*, New York: Springer-Verlag, 1984.
[43] A. Rosenfeld and G.J. Vanderbrug, "Coarse-fine template matching," *IEEE Trans. Systems, Man, and Cybernetics*, 1977, *7*, 104-107.
[44] H. Samet, "A tutorial on quadtree research" in *Multiresolution Image Processing and Analysis*, A. Rosenfeld, Ed. New York: Springer-Verlag, 1984, 212-223.
[45] P.A. Sandon, "A pyramid implementation using a reconfigurable array of processors," *Proc. Workshop on Computer Architecture for Pattern Analysis and Image Data Base Management*, IEEE Computer Society Press, 1985, 112-118.
[46] D.H. Schaefer, "A pyramid of MPP processing elements - experiences and plans," *Proc. 18th Int. Conf. on System Sciences*, Honolulu, 1985.
[47] L. Schmitt, "The use of a network representation of visual knowledge in a hierarchically structured vision system," *Unpubl. Ph.D. Diss.*, Dept. of Computer Sciences, University of Wisconsin, 1981.
[48] D.W. Shaw, "The NON-VON Supercomputer," *Comp. Sci. Dept. Tech. Rept.*, Columbia Univ., August, 1982.
[49] -----, "Organization and operation of a massively parallel machine" in *Computers and Technology*, G. Rabat, Ed. Amsterdam: North-Holland, 1986.
[50] 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. London: Academic Press, 1981.
[51] K. Sloan, "A system for world model driven recognition of natural scenes," *Unpubl. Ph.D. Diss.*, Dept. of Computer Science, Univ. of Pennsylvania, 1977.
[52] S.R. Sternberg, "Cytocomputer real-time pattern recognition." Paper presented at *Eighth Pattern Recognition Symp.*, National Bureau of Standards, 1978.
[53] Q.F. Stout, "Sorting, merging, selecting, and filtering on tree and pyramid machines," *Proc. Int. Conf. on Parallel Processing*, 1983, 214-221.
[54] -----, "An algorithmic comparison of arrays and pyramids," in *Evaluation of Multicomputers for Image Processing*, L. Uhr, K. Preston, S. Levialdi, M.J.B. Duff, Eds. London: Academic Press, 1985.
[55] S.L. Tanimoto, "Pictorial feature distortion in a pyramid," *Comp. Graphics Image Proc.*, 1976, *5*, 333-352.
[56] -----, "Regular Hierarchical Image and Processing Structures in Machine Vision," in *Computer Vision Systems*, A. R. Hanson and E. M. Riseman, Eds. New York: Academic Press, 1978, 165-174.
[57] -----, "Towards hierarchical cellular logic: design considerations for pyramid machines," *Computer Science*

Dept. Tech. Rept. 81-02-01, Univ. of Washington, 1981.

[58] -----, "A pyramidal approach to parallel processing," *Proc. 10th Annual Int. Symposium on Computer Architecture*, Stockholm, 1983, 372-378. (a)

[59] -----, "Algorithms for median filtering of images on a pyramid machine," in *Computing Structures for Image Processing*, M.J.B. Duff, Ed. London: Academic Press, 1983, 123-141. (b)

[60] -----, "A hierarchical cellular logic for pyramid computers," *J. Parallel and Distributed Computing*, 1984, *1*, 105-132.

[61] -----, "An approach to the iconic/symbolic interface," in *Integrated Technology for Image Processing*, S. Levialdi, Ed. London: Academic Press, 1985, 31-38.

[62] S.L. Tanimoto and A. Klinger, Eds., *Structured Computer Vision: Machine Perception Through Hierarchical Computation Structures*, New York: Academic Press, 1980.

[63] J.M. Tenenbaum and H.G. Barrow, "IGS: a paradigm for integrating image segmentation and interpretation," *Proc. 4th Int. Joint Conf. on Pattern Recognition*, 1976, *4*, 504-513.

[64] L. Uhr, "Layered "recognition cone" networks that preprocess, classify, and describe." *Preprints of Conf. on Two Dimensional Digital Signal Processing*, University of Missouri, 1971.

[65] -----, "Layered "recognition cone" networks that preprocess, classify, and describe." *IEEE Trans. on Computers*, 1972, *21*, 758-768.

[66] -----, "Describing, using "recognition cones." " *Proc. 1st Int. Conf. on Pattern Recognition*, Washington, 1973.

[67] -----, ""Recognition Cones" that perceive and describe scenes that move and change over time." *Proc. 4th Int. Joint Conf. on Pattern Recognition*, San Diego, *4*, 1976.

[68] -----, ""Recognition cones" and some test results; the imminent arrival of well-structured parallel-serial computers; positions, and positions on positions," in *Computer Vision Systems*, A. Hanson and E. Riseman, Eds. New York: Academic Press, 1978, pp. 363-372.

[69] -----, "Chapter 1: Parallel-serial variable resolution perceptual systems," in *Structured Computer Vision*, S. Tanimoto and A. Klinger, Eds. New York: Academic Press, 1980.

[70] -----, "Converging pyramids of arrays," *Proc. Workshop on Computer Architecture for Pattern Analysis and Image Data Base Management*, IEEE Computer Society Press, 1981, 31-34.

[71] -----, "Pyramid Multi-Computer Structures, and Augmented Pyramids," in *Computing Structures for Image Processing*, M.J.B. Duff, Ed. London: Academic Press, 1983, pp. 95-112.

[72] -----, *Algorithm-Structured Computer Arrays and Networks: Architectures and Processes for Images, Percepts, Models, Information*. New York: Academic Press, 1984.

[73] -----, "Augmenting pyramids and arrays by embossing them into optimal graphs to build multicomputer networks," in *Parallel Integrated Technology for Image Processing*, S. Levialdi, Ed. London: Academic Press, 1985, 19-31. (a)

[74] -----, "Pyramid Multi-Computers, and Extensions and Augmentations," in *Algorithmically Specialized Parallel Computers*, L. Snyder, L.H. Jamieson, D.B. Gannon, H.J. Siegel, Eds. New York: Academic Press, 1985, 177-186. (b)

[75] -----, "Multiple image and multi-modal augmented pyramid networks," in *Intermediate Level Image Processing*, M.J.B. Duff, Ed. London: Academic Press, 1986, 127-145. (a)

[76] -----, "Constructing multi-level multi-computer networks," in *Evaluating Multi-Computers for Image Processing*, L. Uhr, K. Preston, S. Levialdi and M.J.B. Duff, Eds. New York: Academic Press, 1986. (b)

[77] -----, *Multi-Computer Architectures for Artificial Intelligence*, New York: Wiley, 1986. (c)

[78] -----, *Massively Parallel Hierarchical Pyramid Multi-Computers for Perception*, New York: Academic Press, 1987. (in press)

[79] L. Uhr and R. Douglass, "A parallel-serial recognition cone system for perception," *Pattern Recognition*, 1979, 11, 29-40.

[80] L. Uhr, J. Lackey and M. Thompson, M., "A 2-layered SIMD/MIMD Parallel Pyramidal "Array/Net"," *Proc. Workshop on Computer Architecture for Pattern Analysis and Image Data Base Management*, IEEE Computer Society Press, 1981, 209-216.

[81] S.S. Wilson, "The PIXIE-5000 - a systolic array processor," *Proc. Workshop on Computer Architecture for Pattern Analysis and Image Database Management*, IEEE Computer Society Press, 1985, 477-483.

I.P. HIERARCHICAL SYSTEMS: ARCHITECTURAL FEATURES

V. Cantoni
Dipartimento di Informatica e Sistemistica
Universita' di Pavia
Via Abbiategrasso 209
27100 Pavia, ITALY

Abstract

In this paper the nature of image understanding tasks is outlined and the network computer, multicomputer, and multiprocessor systems based on a hierarchical structure, which have been developed for these tasks, are reviewed.

Finally, the architectural features of the PAPIA machine are presented in details together with the applications that we are developing on this system.

Introduction

In order to satisfy the continuous growth and evolution of computational demands, the search for higher and higher computing speeds has proceeded ceaselessly. Historically, the first approach, pursued as a means of obtaining higher computation speed, has been technological. However, high computer performance depends not only on using faster and more reliable hardware device, but also on the architecture and processing techniques. The main advantage of new architectures lies in a better matching with the data structure and/or with the computational structure.

NATO ASI Series, Vol. F 25
Pyramidal Systems for Computer Vision
Edited by V. Cantoni and S. Levialdi
© Springer-Verlag Berlin Heidelberg 1986

In the area of image processing and computer vision the data structure and the computational structure change as shown in figure 1, in which the heterarchical nature of four processing stages are highlighted:

- preprocessing - the goals are enhancement, restoration, preparation of the image in view of an automatic analysis.

It corresponds to image to image transformation, then the data structure is fixed: the 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, etc.); global operations (digital transforms like Fourier, Hadamard, Haar, etc.). Usually in every point of the image, the same sequence of instructions is applied (no data dependent computations are required), and a small local subarray is involved.

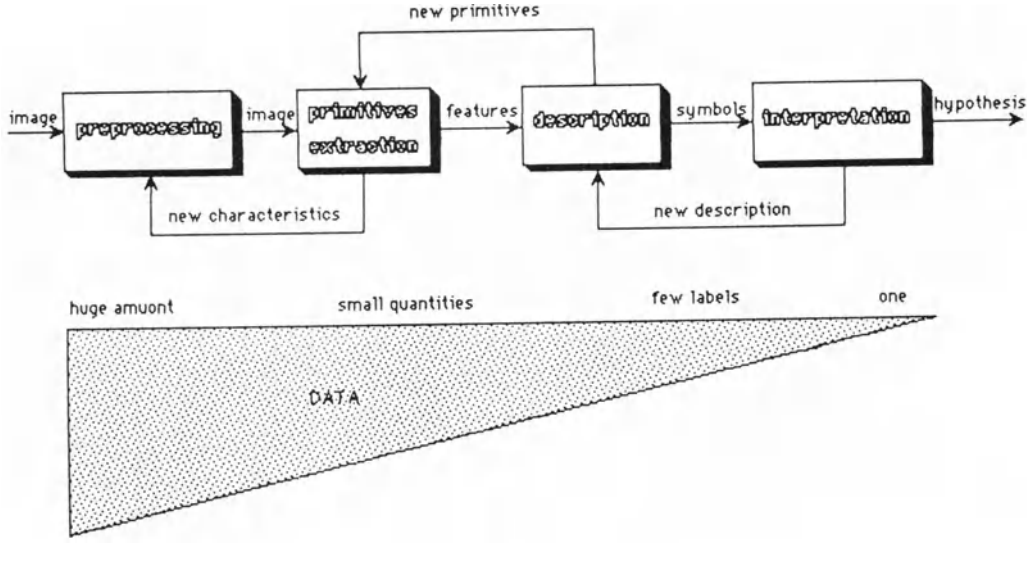


Fig. 1 - Block diagram of the IP process.

- primitives extraction - the goal is the extraction of salient features and pertinent primitives.

It corresponds to an image to features transformation, hence, the data structure changes.

From the computational point of view the segmentation of the image is required; and then, sometimes via basic operations like MAT, skeletonization, edge following, etc., some features are extracted on the base of the grey level statistics (histogramming, co-occurrences, etc.) and other parametrization techniques (moments, shape descriptors, etc.). It is worth to point out that often the operations required at this step realize regional processes a priori not diameter limited. By all means, at this level the processing control is in general again predominately parallel.

- symbolic description - working on the salient features a more compact description to explicit the relations among the primitives is developed.

The final data structure of this step is usually a tree or a graph or a semantic network, in which nodes represent symbols at higher levels of abstraction respect to the original features.

From the computational point of view, at the more abstract levels, algorithms include often inherently serial components.

- interpretation - the final goal is to understand the image contents working on the symbolic description and its context knowledge. The objects included are labelled and identified as well as the relations among the objects.

In this part the data structure refers to the problem of knowledge representation. A correspondence must be find between the symbolic descriptions and an explicit or implicit model of the scene to analyze.

From the computational point of view high flexibility is required to structure, to coordinate and to integrate the different processes that can be applied concurrently. Often in this phase some complementary informations from the

previous stages are needed. In order to perform new measurements, or change parameters, feedback or backtracking processes have to be implemented once a partial interpretation occurred.

A variety of architectural paradigms have been proposed to accomplish these four steps in a hierarchical way: by following a bottom-up approach in which control flows from low level image processing to high level image processing (each step performs its processing or manipulating tasks independently, and passes its output to the next stage); or by following a top-down approach, where the operation is goal oriented and the exchanges are from high level towards low level image processing; or by a more general, heterarchical, approach as described in the last processing stage, and that is shown in figure 1. In this last approach inconsistencies or ambiguities are detected and solved by feedback effects to the previous steps.

Hierarchical architectures for image processing

As it has been quoted above high speed-up factors can be obtained by matching the architecture to the data structure to process, and/or to the computational structure. However, the data structure changes from the "large", fixed array of data of the first steps, to "small", flexible structure of the high level steps. The computational structure is varying as well, the first step deals with the following features: image to image transformations; identical operation applied to all pixels; the computation involved for one pixel requires only the analysis of its own local context. The last class deals with data dependent (asynchronous) operations, sometimes intrinsically sequential and recursive, on flexible data structures.

The SIMD structure is of course well suited for the low level steps that require data independent, synchronous, operations; however, also among these steps there are tasks for which different structures would be preferable. The MIMD structure is not suited for synchronous operations but is more suited for the high level processing steps, which are, usually, non regular and asynchronous.

For these reasons several hierarchical solutions have been proposed based on SIMD, Multi-SIMD, MIMD paradigms, sometimes also mixed in various ways. In what follows a taxonomy is presented in order to introduce some families of machines (see fig. 2). The taxonomy is based on two levels: the former

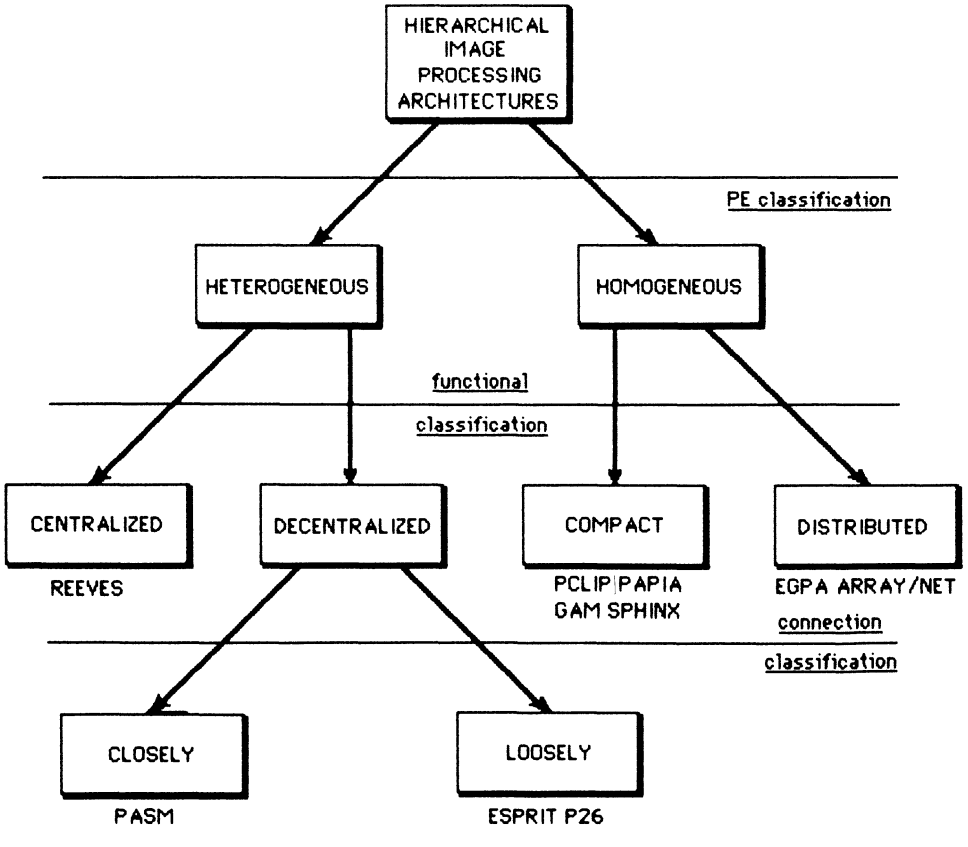


Fig. 2 - Classification scheme of the hierarchical systems.

considers the homogeneity of the processor elements (PEs) or processor units (PUs), the latter is based on the ways of coupling the parts and the interconnection networks among processors.

Five families are identified:

- heterogeneous/centralized - in the first conventional solution, the machine is composed of two parts: a single SIMD processor is used for low level image processing and a separate MIMD for the high level processing part. The two parts are physically different and usually linked by a single bus. This kind of connection do not ease information exchanges between the two subparts, but, due to the independence, separate implementation and debugging can be considered.

In fig. 3 is shown (from [1]) a block diagram of a system belonging to this family. The loose interconnection between SIMD and MIMD parts permit to achieve easily the best features of both the independent components.

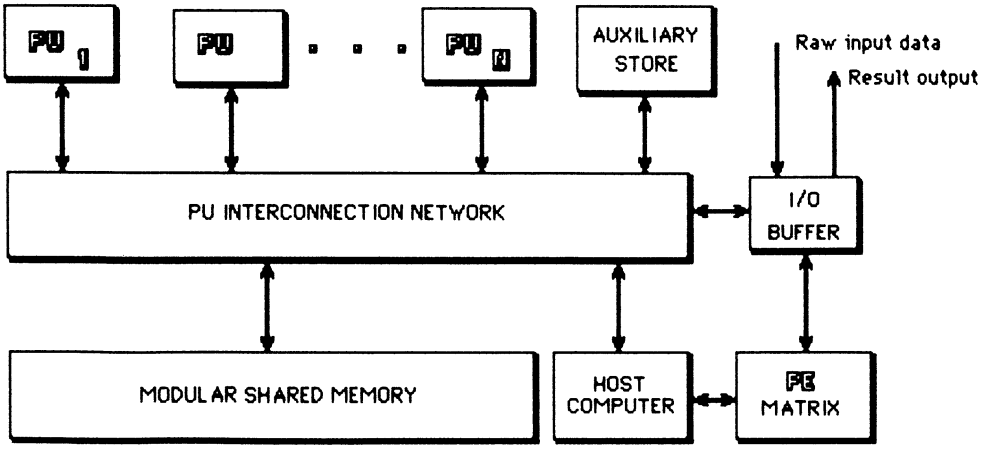


Fig. 3 - Synopsis of an heterogeneous machine composed of two separated MIMD - SIMD subparts.

- heterogeneous/closely distributed - a SIMD sub-system is devoted to each PU of the MIMD structure so that the two parts are not distinct any more. The exchanges in this case are easier, but some overheads, respect to a standard SIMD machine, may arise in low level processing when all the SIMD sub-systems are working in tight "orchestrated" mode, and neighboring access is needed.

The PASM machine which has been developed by Siegel et al.[2] at Purdue University can be considered belonging to this family. PASM is conceived to consist of 1024 processing units (PUs) organized in 16 groups; each group has its own control unit as shown in fig. 4. Adjacent groups may be dynamically configured to behave as a single SIMD system; presumably, for low level image processing tasks, the system would be configured as a single SIMD system of 1024 PUs. The interconnection network among the groups is of the permutation type.

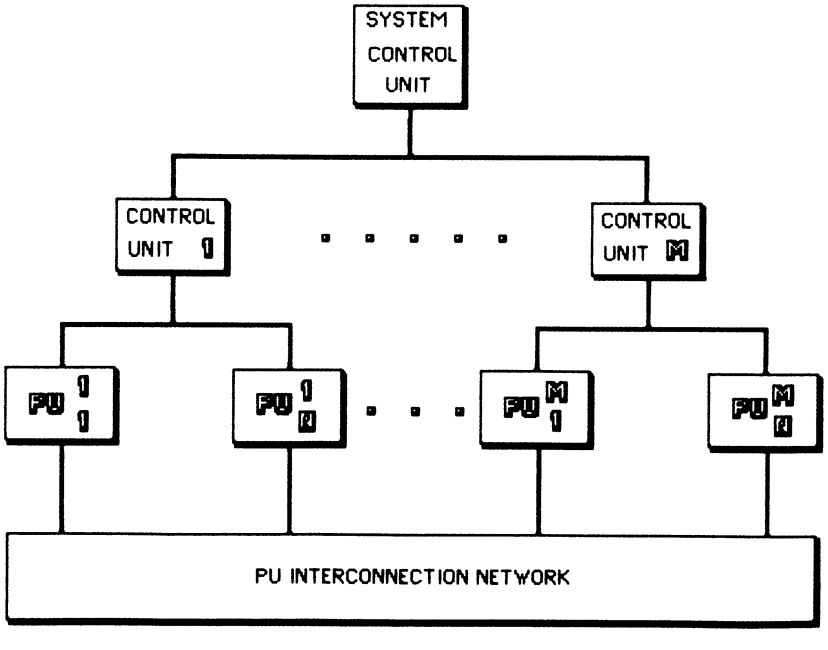


Fig. 4 - Block diagram of the PArtitionable SIMD/MIMD system.

- heterogeneous/loosely distributed - the two sub-systems are physically distinct and linked through as many buses as processor units in the MIMD part. Each bus is supplying information of a subset of SIMD PEs. Several buffers are indeed necessary between the SIMD structure which exchanges its data in synchronous mode and the MIMD structure which works asynchronously. In any case, the parallelization of communication buses increase the capability of exchanging in spite of an higher number of buses and of interface complexity.

A paradigm for this family is shown in fig. 5. In this case each PU of the MIMD part is linked to four near neighbors in a mesh topology. Each processor of the MIMD component

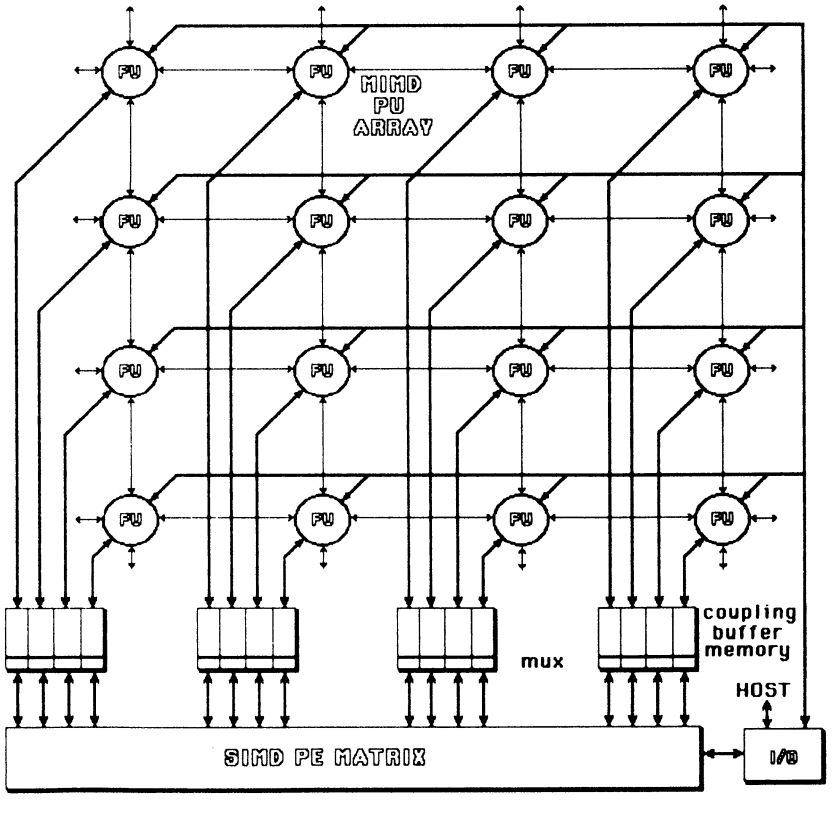


Fig. 5 - Paradigm of an heterogeneous system composed by a SIMD and a MIMD subparts loosely coupled.

collects several PEs of the SIMD part and the data parallelism is usually different: hence an interface with multiplexing and buffer capabilities is needed. These buffers (coupling memories in fig. 5) can be seen as external memories for both the MIMD and SIMD parts, thus they would be dual ported.

- homogeneous/compact pyramid - this class corresponds to that of the Multi-SIMD machines in which several layers of identical PEs work autonomously in SIMD mode. The approach is to set up systems using the largest number of processors (compatible with economical and technological reasons) operating in parallel, by simplifying the single processing unit (usually in these cases bit serial arithmetic is adopted). Some problems arise because, usually, only a small portion of the image can be processed simultaneously, so the approach is to distribute the image to the PEs one block at a time. Performing local operations, difficulties arise in the block border processors, and in the case of iterative local operations, the useful part of the block array propagates inward at every operation.

Among the systems that belong to this family can be quoted (in chronological order): PCLIP[3], PAPIA[4], GAM[5], SPHINX[6].

As an example of machine, belonging to this most popular class, the PAPIA system will be described in detail in the sequel.

- homogeneous/distributed pyramid - the alternative solution is to include (usually) a small number of identical, powerful PUs arranged hierarchically such as to represent a pyramid. These machines belong to the general class called network computers. In this case, for low level image processing, due to the moderate number of PUs, a suitable approach is that of distributing the PUs of the base throughout the image. In this way each PU deals with a portion of the image. Obviously, an overhead due to the border problem and to the coordination of the PUs must be taken into account also in this case.

Referred to the previous solution, generally speaking, it can be said that something is gained for high level image processing capability at expense of the low level processing power (note that the access to the near neighbors of each pixel become serial and, for many low level tasks in which local operators are used, this is a serious bottleneck).

This architectural paradigm can be used in Multi-SIMD mode as well as in MIMD mode. To this last family belong the Uhr's Array/Net[7], and the EGPA[8] shown in fig. 6.

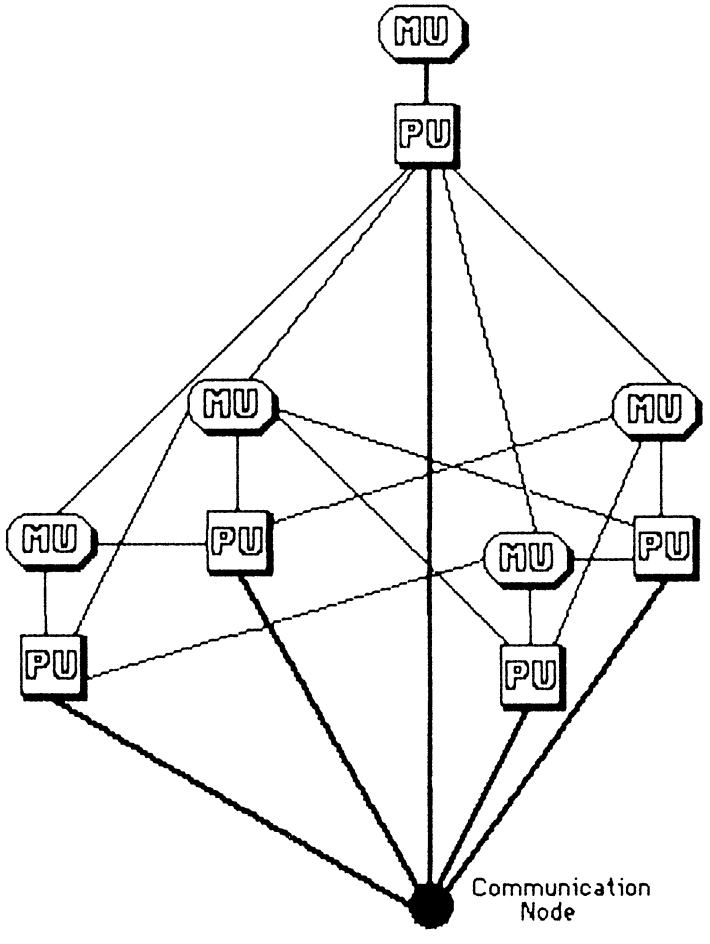


Fig. 6 - Block diagram of the Erlangen General Purpose Array machine.

The PAPIA structure

In 1981 an italian national* research program for the construction of an image image processing system was started. After a first phase devoted to the analysis and comparison of suggested and existing systems and to the definition of a set of benchmarks for the evaluation of the performances of these machines, a new system has been defined. Referring to the taxonomy previously introduced this system belongs to the family of compact pyramids of homogeneous processors (see Fig. 7) and has been called PAPIA which is the acronym of Pyramidal Architecture for Parallel Image Analysis.

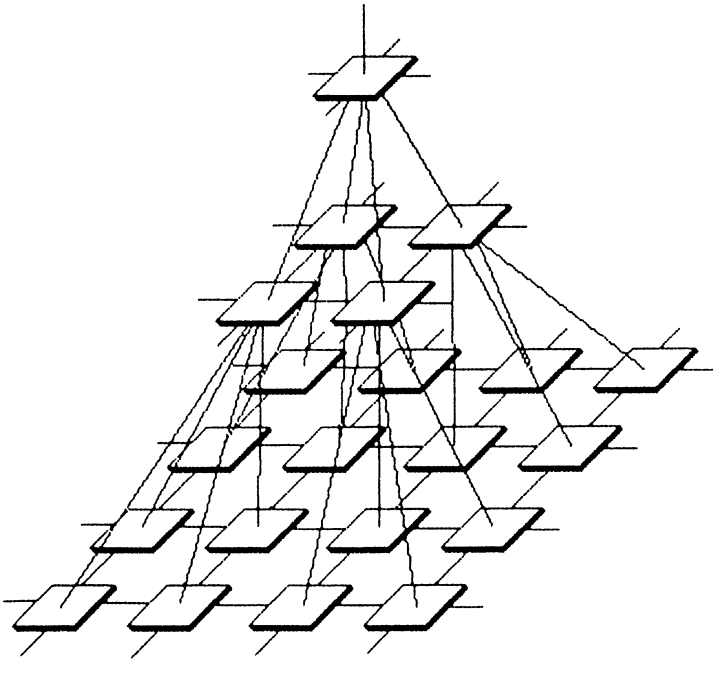


Fig. 7 - Scheme of a compact pyramidal machine (like PAPIA).

---

* The main research groups involved are from the Universities of Pavia, Rome, Milan and Palermo and from the Istituto di Fisica Cosmica ed Applicazioni dell'Informatica (Italian National Research Council) of Palermo.

In what follows, the main architectural features of the system and some applications on which this machine can be exploited, will be highlighted.

Topology and near neighbor access modes.

The PAPIA machine has been conceived as a Multi-SIMD system in which each layer is in principle independent: different instructions may be executed (simultaneously) on different planes (actually, in the first prototype this is true for each couple of planes).

The processor interconnection pattern is the same for all planes, and two kinds of interconnections have been implemented: a "horizontal" one which relates to communication among four brothers of the same layer; and a "vertical" one among PEs of different layers, more precisely with four sons in a preceding plane and with one processor (the father) on a successive plane (see Fig. 8).

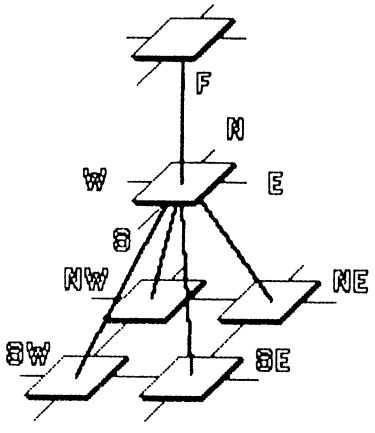


Fig. 8 - Near neighbor connections

Hence, two disjoint operational modes are considered: the horizontal one inside the planes; and the vertical one among different planes.

Any subset of the two near neighbor families may be selected by a gating technique, and an arbitrary boolean function on the enabled neighborhood values may be computed within one clock cycle.

Moreover, the planes are toroidally connected from the last column to the first one and from the last row to the first one; and a square of border registers are included in order to facilitate the computations when the image is larger than the array of PEs considered (in these cases, with compact pyramidal machines, the image to processor distribution mode is usually adopted).

Capabilities of the elementary processor

As all the machines belonging to the compact pyramid family, PAPIA uses one bit processor, so that it is suitable for boolean operations and for implementing bit serial arithmetic. The functional scheme of the processor is shown in Fig. 9.

The first stage of the processor diagram represents the two blocks of the near neighbors selection, described in the previous paragraph, and the local RAM memory, which is composed of 256 bit in the present prototype.

The second and the fourth stages contain the switching elements that in some cases (S1, S6, and S8) include also some simple boolean operators.

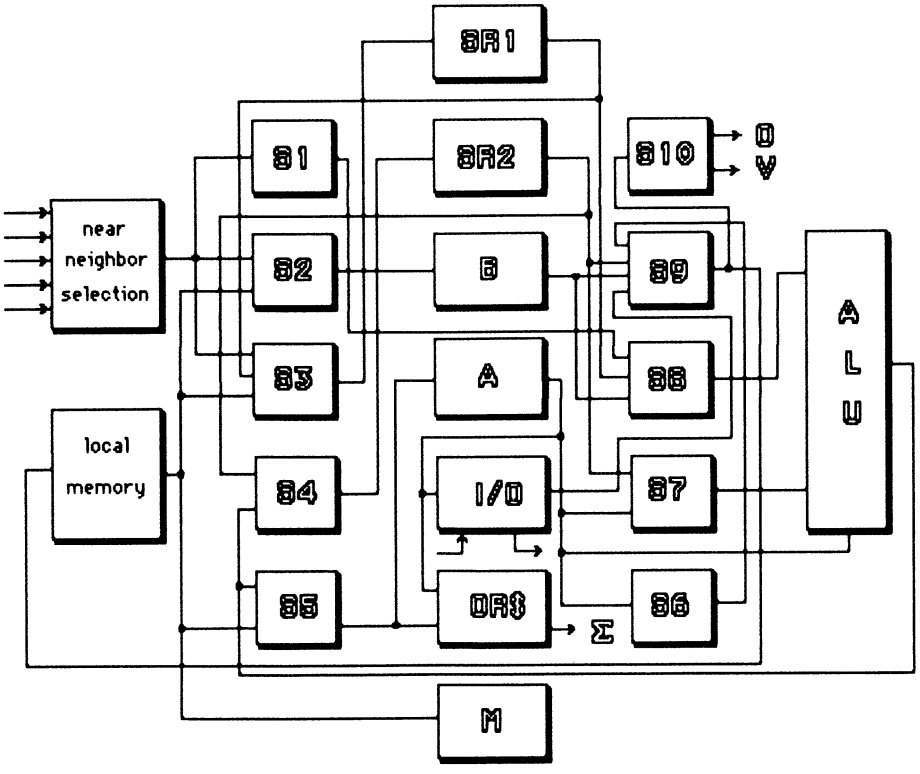


Fig. 9 - Simplified block diagram of the elementary processor of the PAPIA system.

The final stage of the diagram represents the ALU which is composed of: a boolean processor; a full adder with an extra register C for the carry bit and of a comparator circuit.

The third stage of the diagram contains the operative registers. The first couple consists of two variable length circular shift registers SR1 and SR2, which can be used as inputs to the ALU. SR2 works as an accumulator and SR1 can be used to supply the neighboring values. In this way, local operations can be accomplished without storing partial results or refeeding by the neighbors. The comparator of the ALU works between the contents of these two registers.

All the other registers are single bit, master/slave. The two main registers A and B contain respectively: the present status of the corresponding cell, which is broadcasted to the near neighbor class (horizontal or vertical), and the boolean function of the inputs coming from the enabled neighbors.

The B register can be used by means of the S1-S9 path as a shift register distributed in the pyramid. In this way, by an auxiliary clock, a fast long communication among non adjacent PEs can be performed.

Global features and I/O capability

As previously quoted, the pyramid can follow the Multi-SIMD mode in which each layer (or couple of layers in the present prototype) works autonomously. However, also a natural SIMD mode has been considered, in this case a subset of layers can be activated by means of a plane masking register.

Moreover, a PE masking register (in Fig. 9 represented by the M block) enables the selection of a subset of PEs among the active ones. This feature provides the capability to implement data dependent algorithms.

The OR$ register in Fig. 9 belongs to a circuitry (an OR tree) that realizes a global test on single layer enabled PEs. Global tests on the image content are achieved by copying the status register into OR$. This feature is exploited as well in recursive computations in which stable data configurations

must be detected. In these cases the exclusive OR between the present and past states of active PEs is supplied to the OR tree circuitry.

Loading/unloading of image blocks into the PE array is performed by means of the I/O register. These registers are connected in column wise order and are synchronized by the second clock of the system. This allows to adjust the speed to the one of the acquisition device and to overlap completely processing and loading/unloading of images.

Applications

From the point of view of applications, three main activities are currently going on by the research group of the Pavia University on the PAPIA machine:

- implementation of a PDE solver - multi-grid methods have been shown to be an effective means of solving a wide family of elliptic boundary value problems[12]. Standard multi-grid methods can be easily implemented on the PAPIA machine. In fig. 7 a functional block diagram of the "V" cycle of this method is shown. In a first phase the left arm of the cycle is performed by mapping iteratively data on reduced grids that correspond to the different layers of the pyramid.

A solution is found, with fast convergence because of the small amount of data, on one of the higher layers of the pyramid. Then, right arm of the diagram, re-projecting, by interpolation, on the successive layers (to provide the initial condition for the refined solution which is achieved usually by means of few recursive steps in each new layer) the final solution is obtained on the pyramid base.

- implementation of planning strategies - the approach described above can be generalized for many other image processing tasks. A solution of the problem is approached at

the higher levels and then refined successively step by step towards the base. In this way the linear scaling assumption on the maximum speed-up achievable (on the base of the number of processors) can be overcome [13]. For many image processing tasks a particularly useful tool for implementing planning strategies seems to be the two-dimensional Haar transform[14].

- implementation of objects recognition techniques - the generalized Hough transform has been proved to be very effective when the contour pixels of the objects can be labeled on the base of local context characteristics[15], so that each local instance contributes with its own peculiarity to the recognition process. A solution has been developed which permits to label edge points in few iterations on the multi-resolution images of the pyramid (used in SIMD mode)[16]. After the labelling phase the recognition process can be performed on the base of a simple decision theoretic approach.

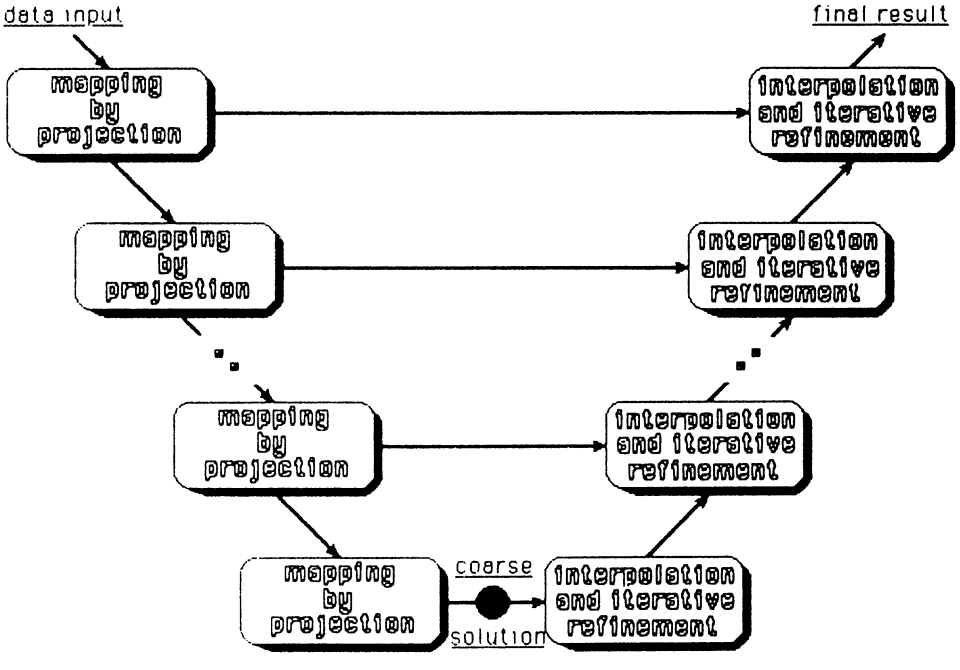


Fig. 10 - The V cycle of PDE solver by multigrid methods.

Conclusions

In the last five years several research groups developed various computervision systems based on hierarchical or heterarchical architectures. In this paper a taxonomy of these machines, aiming to pointout the main architectural features investigated up to now, has been introduced.

The most popular type of these architectures, the compact pyramid, has been described in more details by means of a particular implementation.

Up today, all these machines have been confined to the research institutions area, but they are now becoming ready for the realm of industry.

References

[1] A. P. Reeves, "Parallel Computer Architectures for Image Processing", Computer Vision, Graphics, and Image Processing 25, 68-84 (1984).

[2] H. J. Siegel, "PASM: A Reconfigurable Multi-Minicomputer System for Image Processing", in Languages and Architectures for IP, M. J. Duff and S. Levialdi Eds, Academic Press, New York, 257-266 (1981).

[3] S. L. Tanimoto, "A Hierarchical Cellular Logic for Pyramid Computers", Journal of Parallel and Distributed Computing 1, 105-132 (1984).

[4] V. Cantoni, M. Ferretti, S. Levialdi, and F. Maloberti, "A Pyramid Project Using Integrated Technology", in Integrated Tecnology for Parallel Image Processing, S. Levialdi Ed., Academic Press, London, 121-132 (1985).

[5] D. H. Schaefer, G. C. Wilcox and V. J. Harris, "A pyramid of MPP processing elements - experiences and plans", Proc. 18th Annual Hawaii International Conference on System Science, Vol. 1, 178-184 (1985).

[6] F. Devos, A. Merigot, and B. Zavidovique, "Integration d'un processeur cellulaire pour une architecture pyramidale de traitement d'image", Revue Phys. Appl. 20, 23-27 (1985).

[7] L. Uhr, J. Lackey, and L. Thompson, "A 2-layered SIMD/MIMD Parallel Pyramidal 'Array/Net'", Proc. Work. on Computer Architecture for Pattern Analysis and Image Data Base Management, IEEE Computer Society Press, 209-216 (1981).

[8] G. Fritsch, "Memory-Coupled Processor Arrays for a Broad Spectrum of Application", Lecture Notes in Physics, Springer-Verlag, 158-177 (1984).

[9] V. Cantoni, M. Ferretti, S. Levialdi, and R. Stefanelli, "PAPIA: Pyramidal Architecture for Parallel Image Analysis", Proc. 7th Symp. on Computer Arithmetic, Urbana IL, 237-242 (1985).

[10] V. Cantoni et al., "A Pyramid machine for Image Analysis", Proc. SPIE 8596 Conf. on Architectures & Algorithms for Digital Image Analysis, Cannes, (1985). In Press.

[11] V. Cantoni and S. Levialdi, "PAPIA: a case history", in Massively Parallel Hierarchical Pyramid Multi-Computer For Perception, L. Uhr Ed, Academic Press, (1986).

[12] D. Gannon, "On the Structure of Parallelism in highly Concurrent PDE Solver", Proc. 7th Symp. on Computer Arithmetic, Urbana IL, 252-259 (1985).

[13] C. Hewitt and H. Lieberman, "Design Issues in Parallel Architectures for Artificial Intelligence", A.I. Memo No. 750 (1983).

[14] L. Carrioli, "A Pyramidal Haar Transform Implementation", in Image Analysis and Processing, V. Cantoni, S. Levialdi and G. Musso Eds, Plenum, (1986), in press.

[15] V. Cantoni, L. Carrioli, "Structural shape recognition in multiresolution environment", submitted to Signal Processing.

[16] V. Cantoni, S. Levialdi, "Contour Labelling by Pyramidal Processing", in Intermediate level image processing, M. J. Duff Ed, Academic Press, 179-188 (1986).

GENERAL PURPOSE PYRAMIDAL ARCHITECTURES

Gerhard Fritsch
Universität Erlangen-Nürnberg
Department of Computer Science (IMMD)
D-8520 Erlangen, F.R. Germany

Abstract:

Large problem classes in natural and engineering sciences demand high computational speed and big memory space. These requirements can be satisfied by high performance multiprocessor systems with appropriate architecture such as to allow for efficient mapping of the problem structure onto the multiprocessor structure. Because of a broad variation of the computational parameters of the problems a flexible general purpose computer architecture is needed. Distribution of different functions onto the multiprocessor system such as user's program execution, operating system functions, input/output etc. can be achieved through space-sharing. This favors hierarchically structured architectures, in particular pyramidal systems.

1. INTRODUCTION

The development of the computer between the forties and the present time has essentially been the development of the von Neumann machine to higher and higher performance. This has been possible through the fast technological progress, from the radio tube to the transistor and to the integrated circuit. Only recently, the increase in speed of conventional computers (monoprocessor) slows down. Although computational speed will grow in future through technological innovations (GaAs, digital optical computer), limiting effects due to physical laws show up (e.g. an electromagnetical signal needs at least 1 ns to travel 30 cm). This fact and new technological possibilities (e.g. inexpensive VLSI microprocessors) have contributed to the appearance of the parallel computer which should be appreciated as a qualitative change in the history of computer architecture. Parallel computers are characterized by pro-

NATO ASI Series, Vol. F 25
Pyramidal Systems for Computer Vision
Edited by V. Cantoni and S. Levialdi
© Springer-Verlag Berlin Heidelberg 1986

cessing and storing data in a distributed and simultaneous way effected by several (or many) components. With parallel computers new organisational elements are used: interprocessor communication, distributed memory management, specialized processors (e.g. I/O-processors) and so on. Nowadays, a broad spectrum of parallel computer architectures exists. There is a variety of specialized fast computing machines, as for instance image preprocessing machines of the SIMD-type (single instruction multiple data). On the other hand, various classes of general purpose or specialized general purpose machines have been developed, e.g. for numerical computations of mathematical problems. Such computers frequently are flexible multiprocessor-systems of the MIMD-type (multiple instructions multiple data). In general, multiprocessors are organized in two or more hierarchical levels. General purpose multiprocessor systems with a pyramidal organisation can be compared, in analogy, to human organisations. At the top overall management and control of the system should be located. Operating system tasks may be performed at intermediate hierarchical levels whereas user's programs run at the lowest levels which will be equipped with the major portion of the total processor-memory resources. Concepts and realisations of such multiprocessor systems and their application are discussed in this paper.

2. APPLICATION PROBLEM STRUCTURE AND MULTIPROCESSOR ARCHITECTURE

2.1. Compute-intensive applications in natural and engineering sciences

Progress in natural and engineering sciences has evolved through a fertile interaction between theory and experiment. On one hand, results obtained in laboratory experiments have demanded theoretical interpretation. On the other hand, theoretical work has often stimulated experimental progress by proposing new experiments or even predicting new experimental results.

With the advent of large computers a third way to increase scientific knowledge has emerged: computational physics, computational chemistry or, more general, computational science (Fig. 1).

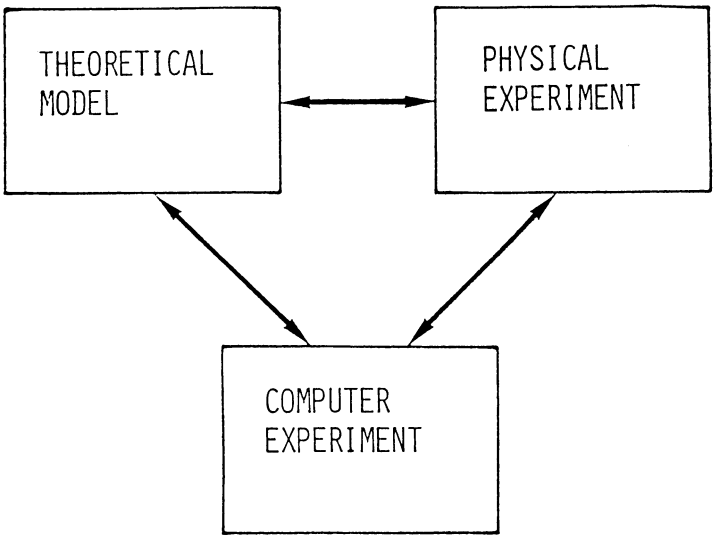


Fig. 1: Simulation of physical phenomena in the computer experiment (computer + simulation program). Traditional interaction between theory and experiment: Experimental results demand theoretical interpretation and theoretical proposals call for experimental verification. The computer experiment substitutes, supplements or extends the laboratory experiment.

Computational physics methods are applied to the study of physical phenomena. The work starts from a theoretical model whose analytical treatment, however rapidly gets unmanageable as the particular phenomenon to be studied becomes more and more complicate. Therefore, numerical methods have to be applied. This often needs high computational power. In the discretization step, say, of a system of partial differential equations (PDEs) which describe a physical phenomenon, space and time continuum is replaced by a special grid and a sequence of time steps. By discretization continuous physical variables are replaced by (2D- or 3D-) arrays of values. By use of some relaxation method we obtain, by iterations, an approximative solution of the mathematical problem. In order to obtain a physically reasonable result from such a "numerical model" the discretization length (width of the numerical mesh) and the time step have to be chosen sufficiently small. Therefore, the demand for computational speed can be enormous. Only with modern large computers refined models can be computed, for example when a 2D-model is replaced by a more realistic 3D-model. By variation of some physical parameters of the model "computer experiments" can be carried out. This is of practical importance in case that laboratory experiments are not feasible (e.g. astrophysics, molecular dynamics) or ex-

pensive (e.g. wind tunnel measurements). The principal step in setting up a computer experiment of a physical phenomenon can be summarized as follows /1/, /2/:

- Description of the phenomenon by a mathematical model (e.g. by a system of partial differential equations, PDE)
- Discretization and linearization of continuous equations (discrete algebraic approximation)
- Implementation of a simulation program.

Numerical simulation has become an important tool for solving application problems in natural and engineering sciences, as fluid dynamics, condensed matter physics (e.g. semiconductor physics), plasma physics, chemistry (e. g. molecular collision processes, surface and colloid chemistry), biological systems, nuclear physics, applied geophysics and others. Most application problems in these areas are described by a system of partial or ordinary differential equations. The numerical treatment (linearization, discretization) leads to large systems of linear algebraic equations. The computational task of such numerical problems varies in a wide range according, to the particular applications and the complexity and accuracy of the used model.

Another large problem class with inherent parallelism is image processing. Various parallel computer architectures have been proposed, /3/, /4/, /5/, /6/, /7/ and others. Low level image processing demands fast special parallel computers of the SIMD-type (single instruction multiple data). For intermediate (feature extraction and classification) and high processing levels (image analysis, understanding, decision) more flexible MIMD-type (multiple instructions multiple data) multiprocessor systems are appropriate. The concept of the processing cone for image processing problems is wellknown /8/. At the lowest level (preprocessing) many but simple and identical operations have to be carried out on huge data amounts. Towards higher levels the data amounts to be processed decrease considerably whereas the complexity of the operations augments. Matching the subtask structure to an appropriate multiprocessor architecture needs a detailed analysis of the processing and data transfer costs, load balancing of the multiprocessor system etc. The architectural requirement favors a hierarchical architecture, in particular pyramidal ones /9/, /10/.

Another problem class comprises non linear multidimensional optimization problems. A typical application is the optimization of complex technical systems (e.g. maximization of the output of a chemical reactor /11/). At the highest task level an optimization strategy (decision level) is effected. At the next lower level the mathematical model of the technical system can be computed and the objective function calculated. At the lowest level highly compute-intensive auxiliary calculations (e.g. numerical integration) are being performed. Such a pyramidal computational structure can be efficiently mapped on an appropriate pyramidal architecture.

The production of VLSI circuits demands reliable simulation, in order to predict circuit operation and performance and to verify the correct behavior of the designed circuit. In the near future, this will be of growing importance for the production of integrated circuits with higher and higher density of components on the chip. There are several simulation levels which differ in "granularity", that is in the degree of detail to be simulated. The following Fig. 2 depicts a pyramidal representation of the simulation levels.

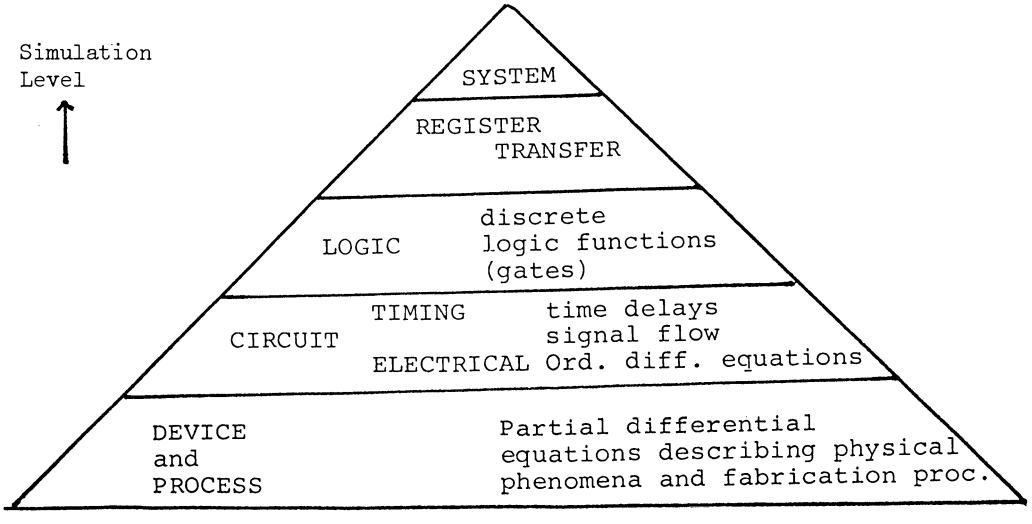


Fig. 2: Simulation cone of VLSI circuit simulation.

The area at each level is qualitatively "proportional" to the expected computational cost. The simulation levels can be characterized as follows:

System Simulation: High level modeling of system behavior describing the operation and performance of e.g. computer components (arithmetic-logic units,

RAM, data paths etc.)

Register Transfer and Logic Simulation: Description of standard circuits (AND, OR, NAND etc.) and of larger composed circuits as to their logic behavior, including time-dependent logic behavior.

Circuit Simulation: Description of the electrical behavior of circuits consisting of several single devices (transistors etc.). The simulation comprises DC models and timing models (time dependency of signal levels) and takes into account technology-dependent parameters (CMOS, bipolar). The mathematical model is represented by a system of ordinary differential equations.

Device Simulation: Analysis and optimization of device parameters of the used technology, the production process and layout geometry. The physical phenomena involved are described by partial differential equations (PDE).

Process Simulation: Determination of appropriate values of the process parameters for given device features (geometric, switching time, electrical power and so on). As with device simulation models, the process phenomena (e.g. diffusion process) are described by sets of PDE.

2.2. Architectural Requirements

As the issue of this article is "general purpose architecture" we should look for common problem features of the application classes we are discussing. Large problems from natural and engineering sciences have computational structures which can be characterized as follows:

- Demand for high processing power, large memory capacity and fast I/O data transfer
- High degree of parallelism
- Regular data structures
- Data or task partitioning
- Computational interaction between neighbored data subregions (this is a consequence of local interaction in physical phenomena); little global data transfer
- Need for flexible and fast data distribution because of local particularities in the data field (e.g. data spreading with local refinements)
- Two or more level hierarchy of the subtask structure of complex problems.

These requirements may form a general basis for appropriate architectures. However, designing an efficiently computing multiprocessor system needs some more inputs. For one type of problem, problem parameters can change strongly from one given case to another. The number of operations executed per memory access can vary considerably. For instance, the relaxation operation on each grid point of a numerical mesh (e.g. generated by PDE discretization) can be found between two extreme cases: many operations on few data and few operations on many data. With numerical simulation in aerodynamics discretization of 3D problems can result in grid sizes of more than 10^9 grid points. The number of physical variables related to each grid point can vary between 10 and 30. All variables have floating point values with 32 or 64 bit wordlength so that about 10^{11} byte or more of storage capacity is necessary. If we assume 100 operations per grid point and time step, the total computation amounts to more than 10^{14} arithmetic operations for 1000 time steps.

To cope with these often conflicting demands, the computer designer has to select a suitable multiprocessor interconnection system and find a tradeoff among competing requirements. Furthermore, he can take into account special computational needs, e.g. by using special components in the node processor/co-processor, providing a large number of I/O channels, supporting microprogramming.

Another important feature of the multiprocessor system to be designed is expandability. Because of the users' demand for constantly growing performance, the multiprocessor architecture must be expandable, without increasing the complexity of the interconnection system more than linearly with the total number of processors.

In order to satisfy the computational requirements following "general purpose" /12/, /16/, /27/ features of a multiprocessor architecture are proposed:

- The multiprocessor system is composed of many parallel working processor-memory modules (PMMs), each one with high processing speed and large memory space.
- The interconnection system is characterized by regular and homogeneous 2D- or 3D-arrays of PMMs (regularity as to the interconnection system, homogeneity as to the internal PMM structure)
- The arrays are arranged in hierarchical order, so that user's programs run on low level arrays whereas administrative tasks (higher operating system functions, I/O) and possibly some user's tasks are operated on high level

arrays.

- Neighborhood of each PMM should be restricted (to four PMMs in case of an 2D-orthogonal array), so that local complexity remains constant and global complexity grows proportionally to the total number of PMMs in the array
- Hierarchically arranged arrays of PMMs should be connected "vertically" as to form a hierarchical multiprocessor system ("pyramid").
- The multiprocessor system should be expandable in order to be able to meet user's increasing computational demands.

3. PYRAMIDAL ARCHITECTURES

3.1. Nearest - Neighbor - Multiprocessors Systems

As we have discussed in chapter 2, in many numerical applications there is a natural local interaction. With respect to a discretization grid this demands nearest-neighbor processing structures which can be realized by orthogonal (or hexagonal etc.) grids of processor-memory-modules (PMMs). In 2D- or 3D-processor arrays each PMM has 4 or 6 neighbors respectively. Realistic models use 3D-meshes. Therefore, best data mapping would be achieved on 3D-processor arrays. However, for practical reasons - interconnection complexity, programming techniques - 2D processor arrays are being preferred. 3D data meshes can be computed by 2D processor arrays layer by layer.

Beside the computational tasks performed on the array of worker PMMs, service and supervising functions have to be provided which may be implemented on the very same worker processor array, or on another array or host at higher hierarchical level. With other words, there is a hierarchy of tasks as

- overall control
- operating system functions, including input/output management,
- user problem computation, including local control

which can be assigned to hierarchical levels of a multiprocessor system, in a natural manner.

Thus, an ample variety of architectures can be realized: pyramidal systems of hierarchically ordered processor arrays (e.g. /12/), processor array with supervising host, e.g. /13/, processor array with one PMMs having master function, and others.

Interconnection between PMMs in a processor array can be achieved by bus systems, or by shared memory (with dual or multiports). A famous example for the first interconnection type is the Cm* project /14/ whose communication system relies on three levels of buses. A more recent project, SUPRENUM - supercomputer for numerical applications /15/ uses a similar concept on two levels of busses. At the lowest level, 16 floating point processors (one cluster) are tightly coupled by a cluster bus. Through a communication node, each cluster is connected with the (possibly) fifteen other clusters of the system by a two-dimensional (orthogonal) system bus. The total multiprocessor system is expected to perform 1GFLOPS.

Distributed shared memory (DSM) systems with nearest-neighbor interprocessor communication have open-ended expandability - contrast to bus systems - because of constant local interconnection complexity (each PMM is connected with 4 or 6 neighbors in a 2D- or 3D array respectively). Furthermore, the total "local" intercommunication bandwidth generally is much higher than with bus systems. Examples for DSM arrays are the worker processor array of the EGPA system (Erlangen General Purpose Array) /10/, /16/, /27/, which will be discussed in the next chapter, the EBR-project /28/ and the PAX system/13/, /26/.

3.2. Experimental pyramid system

In order to gain practical experience with multiprocessor systems two experimental systems were designed and built at the University of Erlangen-Nürnberg/ F.R. Germany. "Experimental" means that research work has aimed at the study of architectural issues rather than at high performance use for applications. The EGPA-pilot pyramid (financed by the German Federal Ministry for Research and Technology) was realized in 1978 /12/, /16/. The pilot system consisted of 4 worker PMMs with mutual memory access between neighboring PMMs, and one supervising PMM with access to the memory of the 4 worker PMMs (unidirectional connections). The PMMs were commercially available computers AEG 80/60 (32 bit wordlength, microprogrammable, multiport memory). The operating system was a hierarchical multiprocessor operating system based on the original uniprocessor operating system /16/.

Interprocessor communication was effected via mailbox techniques. The programming system supported user task management and communication between different tasks (by messages or common data). The allocation of data to suitable memory modules was also supported /17/,/18/.

Other multiprocessors were configured from PMMs of the Distributed Reconfigurable Multiprocessor Kit (DIRMU) which is also based on memory-coupling by multiport memories /19/, /20/, /21/, /22/. Each PMM consists of a processor module which contains one Intel 8086/8087 microprocessor (and coprocessor), and of a memory module which is organized as multiport memory. The memory module of each PMM can be accessed by its "own" processor module and neighboring processor modules. A large variety of multiprocessor configurations (array, cube, tree, ring, pyramid and others) can be composed. Thus, an array of 5x5 PMMs and a 21-processors pyramid (Fig. 3) have been built. Other purposes of this experimental multiprocessor are investigations of selfdiagnosis and techniques for fault-tolerance as well as of multiprocessor operating systems /22/. In 1985 the 21-processors pyramid replaced the 5-processors pilot pyramid (Fig. 3).

On these multiprocessor systems a broad spectrum of user problems was implemented: Matrix operations, relaxation, multigrid-methods non-linear optimization, transients of electric power systems, topographical problems. Efficiency values ranged between 90 and 100%. Only problems with high inherent sequential subtask structure yielded lower efficiency (65% with text formatting).

We should add some remarks as to programming DSM multiprocessor systems. From the implementation of user programs on the experimental machines we have learned that the programming technique is essentially the same as for a monoprocessor. The user has to parallelize his problem by task partitioning or more frequently by data partitioning in a suitable way and to associate the subtasks or data portions with the PMMs in a natural way. The aim is to achieve load balancing on the processor array. Each PMM works on its data portion being controlled by usual monoprocessor programs to which only some communication parts have to be added /30/, /34/.

However, it cannot be expected that a compiler could recognize all the interactions involved and generate optimal mapping automatically. Actually, this has to be carried out by the user.

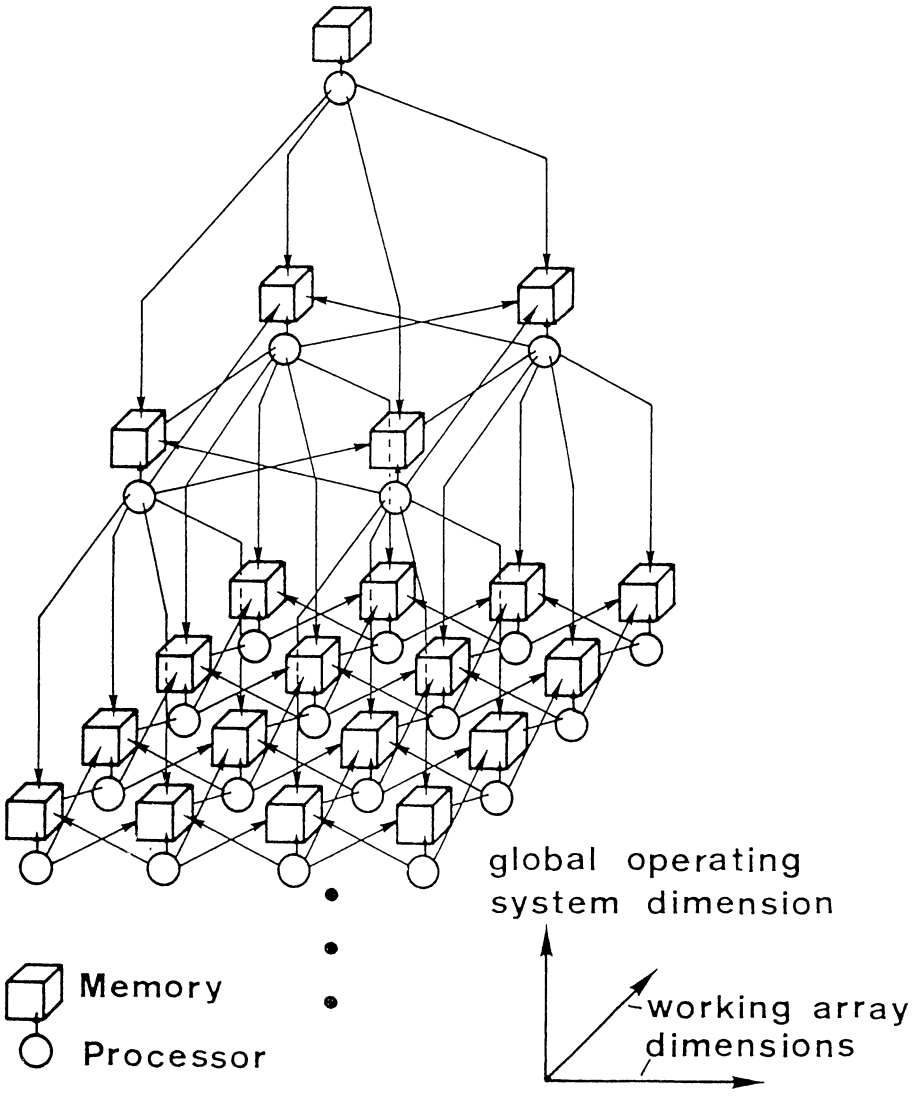


Fig. 3: Erlangen General Purpose Architecture:
21-processor experimental system

3.3. High performance pyramid systems

Powerful computational performance of a multiprocessor system can be achieved both by using a high speed node processor and by connecting a large number of them. Furthermore, the underlying architectural concept strongly affects the overall performance and the cost-performance ratio of the system. In section 2.2 we discussed the requirements for an appropriate architecture. The Erlangen concept /12/, /16/, /27/ for a high performance multiprocessor system meets these requirements. It has been largely based on the results obtained with various multiprocessor projects (EGPA, DIRMU, SYMPOS /31/, MUPSI /32/). The principal features of the EGPA architecture can be summarized as follows:

Tight coupling: The architecture consists of processor-memory-modules (PMM) which are connected in two-dimensional orthogonal grid-like structures. Each processor has access to the memories of the four adjacent PMMs (bidirectional connections) and each memory can be accessed by neighboring processors through a multiport control device.

Restricted neighborhood: By coupling each PMM with four neighbors the local complexity remains constant throughout the processor plane, whereas the global complexity grows proportionally to the number of PMMs in the system.

Regularity and homogeneity: A rectangular array of homogeneous processors is most appropriate for many numerical applications which demand highly regular data-exchange.

Hierarchy: Two or more arrays of PMMs constitute a hierarchical system through "vertical" connections. The administrative and I/O-modules of the global operating system are concentrated in the higher levels, while most of the user tasks are processed in the lowest array. Each processor - except those at the lowest level - has access to the memories of four subordinate PMMs (unidirectional connections). The higher planes of PMMs may be used not only for control purposes but also for data transfer between worker PMMs which are not neighbors. Performing different "tasks" simultaneously in the multiprocessor system can be achieved by space-sharing /29/, which replaces time-sharing. Space-sharing means that different tasks are sharing the processor arrays, whereas each processor is assigned to only one task. The "tasks" can be different user programs or system functions executed at different hierarchical level (user problem, operating system, overall control).

However, the overall performance of the system depends strongly upon the processing power and the memory capacity of the individual PMM. Each PMM may consist of the following parts:

- A conventional microprocessor (e.g. MC 68020, Intel 80386, NS 32 032)
- 4 Mbyte memory (multiport)
- multiport controller
- high speed microprogrammable coprocessor.

The coprocessor is necessary to obtain high arithmetic performance with numerical applications. Due to microprogrammability powerful microprograms can be implemented which execute long sequences of operations, e.g. an entire relaxation step. The overall structure of a powerful coprocessor /33/, /23/ has the following features (Fig. 4):

- Instruction interface to buffer commands from the node processor instruction stream.

- Control units for the addressing unit and the arithmetical units:
They are driven by the parameters from the instruction interface and include microprograms for the algorithms and for the computation of the grid point addresses.

- Arithmetic units should be equipped with high speed buffer memories that are used in an interleaved way by the arithmetic and addressing units. By buffering grid values, the number of accesses to the node's main memories can be reduced drastically because of the locality of the algorithms. As floating point unit the ALU Am 29 325 can be used. The unit is able to execute 32 bit floating point operations internally within 100 ns.

- Addressing units to unload the arithmetic section of the coprocessor from the task of address calculation for the grid points within the node's main memories. Independent address calculation units are needed that fill one data buffer with operands while the arithmetic units work on the other buffer.

Thus, for the individual PMM a performance of 4 MFLOPS can be expected. This performance can also be obtained for scalar operations, since the Am 29 325 is not working in an internally pipelined mode.

As large numerical problems need high memory capacity, the multiprocessor system must be provided with enough parallel I/O-channels to secondary memory. We propose 64 I/O-channels (10 Mbyte/s transfer rate each) which should be

connected to the 64 PMMs located at the second hierarchical level.

With these features, the maximum overall system performance for 256 working processors will be (Fig. 5):

- Computation power 1 GFLOPS
- Main memory capacity 1 GB
- Transfer rate (overall) 640 Mbyte/s

Besides numerical applications, there is a demand for high performance in non numerical applications, as for instance in various fields of artificial intelligence. Diversification - or individualisation - of the node processor through microprogramming allow for efficient adaptation to the requirements of the particular application area /35/.

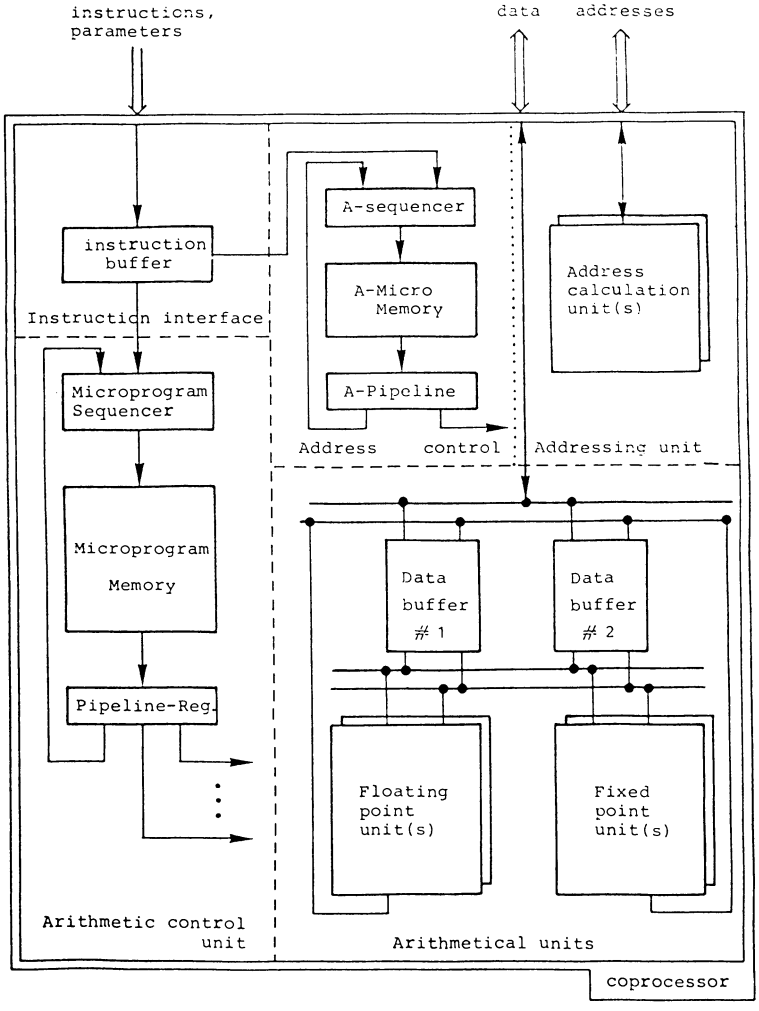


Fig. 4:
Structure of the microprogrammable node co-processor for high speed numerical computation of an High Performance EGPA system (Bode et al. /23/).

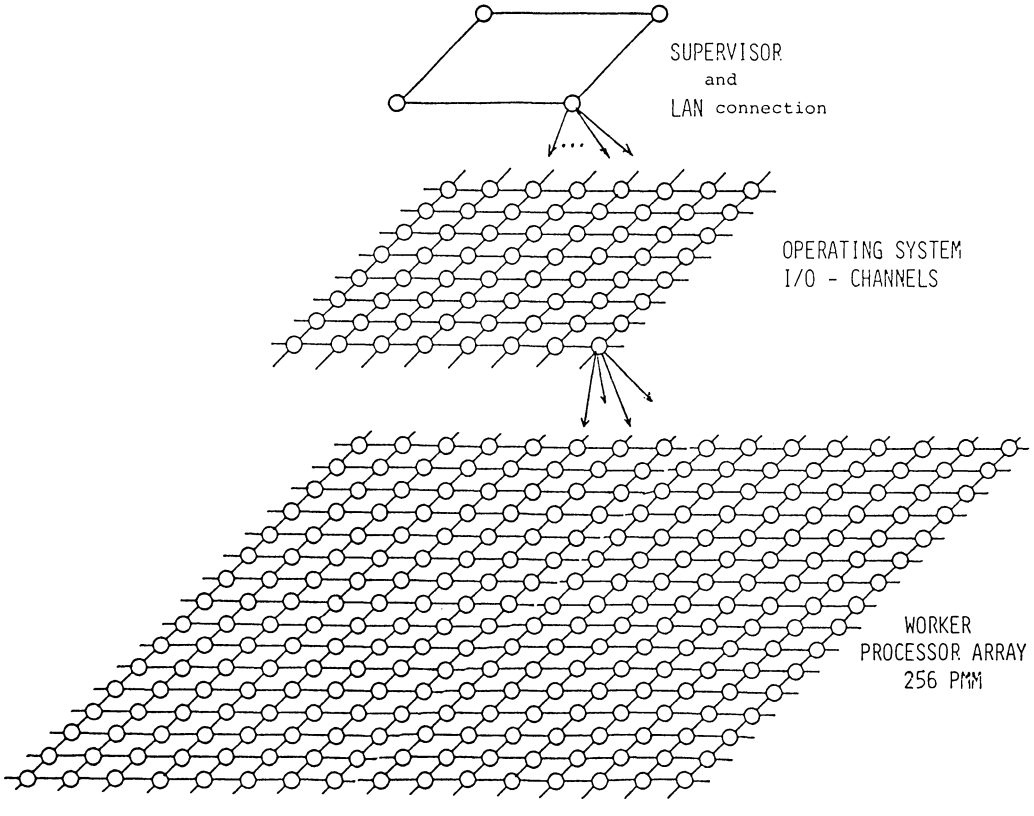


Fig. 5: High performance multiprocessor system:
Erlangen General Purpose Architecture
(Distributed Shared Memory interconnection)

4. CONCLUSIONS

Future increase in computing power will be essentially rendered possible by parallel processing architectures. Physical limits to enhance computational speed only by technological means have shown up. This fact has essentially contributed to the tendency to parallel computers. For general purpose use multiprocessor systems can be built with good flexibility and a maximum theoretical speedup equal to the number of worker processors. The efficiency of multiprocessors depends largely on fitting the problem structure to the multiprocessor structure. By appropriate task partitioning and data partitioning the user problem has to match the processor-memory structure. For a general purpose use rather than for special application the computer architecture must have suitable features that allow for efficient computation of user problems whose parameters vary in a wide range. Fundamental features are regularity and homogeneity of the multiprocessor structure, an effective interconnection network for the processor and memory modules and an hierarchical order of the system components according to their function. The latter can be realized by hierarchical space sharing in the multiprocessor corresponding to the hierarchy of tasks as explained in paragraph 3.1., This suggests pyramidal topologies.

Distributed shared memory (DSM) systems generally have a regular distribution of processing elements and attached memories. So data are stored where they are needed for processing. Furthermore, DSM systems have beneficial properties because of tight coupling between (neighboring) processor-memory-modules, the use of a multiprocessor operating system with a straightforward structure and the ease of user programming.

Due to enormous costs for hardware and software development expandability of multiprocessor systems is an important architectural feature which has important economic implications to further supercomputer developments.

5. REFERENCES

/1/ Hockney, R.W.; Eastwood, J.W.: Computer Simulation Using Particles. McGraw-Hill (1981)

/2/ Rodrigue, G.; Giroux, E.D.; Pratt, M.: Large-scale Scientific Computation Computer 13,11, 65-80 (198/)

/3/ Reeves, A.P.: Parallel computer architectures for image processing. Proc. 1981 Int. Conf. Parallel Processing, 199-206

/4/ Dyer, Ch.R.: A VLSI pyramid machine for hierarchical parallel image processing. Proc. Conf. Pattern Recognition and Image Processing, Dallas Aug. 3-5, 1981, 381-386

/5/ Kushner, T.; Wu, A.Y.; Rosenfeld, A.: Image Processing on ZMOB.IEEE Trans. Comp., Vol. C-31, No. 10, Oct. 1982, 943-951

/6/ Uhr, L.: Converging pyramids of arrays. IEEE Computer Soc. Workshop CAPAIDM 1981, 31-34

/7/ Uhr, L.; Thompson, M.; Lackey, J.: A 2-layered SIMD/MIMD parallel pyramidal "array/net". IEEE Computer Soc. Workshop CAPAIDM 1981, 209-216

/8/ Nagin, P.A.; Hanson, A.R.; Riseman, M.: Region relaxation in a parallel hierarchical architecture. In: M. Onoe, K. Preston, A. Rosenfeld, Real-Time Parallel Computing Image Analysis, Plenum Press 1981

/9/ Cantoni, V. and S. Levialdi: Matching the task to an image processing architecture. 6th Int. Conf. Pattern Recognition, Munich, Oct. 19-22, 1982, 254-257

/10/ Cantoni, V.: Organization of multi-processor systems for image processing. In: Lect. Notes in Physics 196 (Ed. J. Becker and I. Eisele), 145-157, Springer Verlag 1984

/11/ Fritsch, G.: Memory-coupled processor arrays for a broad spectrum of applications. In: Lect. Notes in Physics 196 (Ed. J. Becker and I. Eisele), 158-177. Springer Verlag 1984

/12/ Händler, W.; Hofmann, F.; Schneider, H.J.: A General Purpose Array with a Broad Spectrum of Applications. In: Händler, W.: Computer Architecture, Informatik Fachberichte, Springer Verlag Berlin Heidelberg New York, 4, 311-35 (1976)

/13/ Hoshino, T. et al.: Highly parallel processor array "PAX" for wide scientific applications. Proc. 1983 Int. Conf. on Parallel Processing, Aug. 23-26, 1983, 95-105

/14/ Swan, R.J.; Fuller, S.H.; Siewiorek, D.P.: Cm*- A modular multi-microprocessor. AFIPS Proc. NCC, Vol. 46, 1977, 637-644

/15/ Suprenum, Vorhabensbeschreibung, Gesellschaft für Mathematik und Datenverarbeitung mbH, Inst. f. Math. Grundlagen, St. Augustin/F.R. Germany, Oct. 1985

/16/ Händler, W.; Herzog, U.; Hofmann, F.; Schneider, H.J.: Multiprozessoren für breite Anwendungsgebiete: Erlangen General Purpose Array. GI/NTG-Fachtagung "Architektur und Betrieb von Rechensystemen", Informatik-Fachberichte, Springer Verlag Berlin Heidelberg New York, 78, 195-208 (1984)

/17/ Fromm, H.J.; Hercksen, U.; Herzog, U.; John, K.-H.; Klar, R.; Kleinöder, W.: Experiences with Performance Measurement and Modelling of a Processor Array. IEEE-TC, C-32, 1, 15-31 (1983)

/18/ Hercksen, U.; Klar, R.; Kleinöder, W. and Kneißl, F.: A method for measuring performance in a multiprocessor system. In: Proc. 1982 ACM SIGMETRICS Conf., Seattle, Wa., 77-88 (1981)

/19/ Händler, W.; Rohrer, H.: Thoughts on a Computer Construction Kit. Elektronische Rechenanlagen 22, 1,3-13 (1980)

/20/ Händler, W.; Schreiber, H.; Sigmund, V.: "Computation Structures Reflected in General Purpose and Special Purpose Multi-Processor Systems",

Proc. 1979, Int. Conf. on Parallel Processing, pp. 95-102

/21/ Händler, W.; Maehle, E.; Wirl, K.: DIRMU Multiprocessor Configurations, Proc. 1985 Int. Conf. on Parallel Processing, St. Charles 1985, 652-656. IEEE Comp. Soc. 1985

/22/ Maehle, E.: "Fault-Tolerant DIRMU Multiprocessor Configurations", Computer Architecture Technical Commitee Newsletter, IEEE Computer Society, 51-56, June 1985

/23/ Bode, A.; Fritsch, G.; Henning, W.; Volkert, J.: High performance multiprocessor systems for numerical applications. Proc. First Int. Conf. on Supercomputing Systems, St. Petersburg/Florida, Dec. 16-20, 1985, 460-467, IEEE Comp. Soc. Press 1985

/24/ Fritsch, G.; Kleinöder, W.; Linster, C.U.; Volkert, J.: EMSY85 - The Erlangen Multiprocessor System for a Broad Spectrum of Applications. Proc. 1983 Int. Conf. on Parallel Processing, 325-330 and in: Supercomputers: Design and Applications (K. Hwang, ed.), IEEE Comp. Soc. (1984)

/25/ Gottlieb, A. et al.: The NYU Ultracomputer-Designing an MIMD Shared Memory Parallel Computer", IEEE Transactions on Computers, vol. C-32, Feb. 1983, pp. 175-189

/26/ Hoshino, T. et al.: PACS, a parallel microprocessor array for scientific calculations. ACM Trans. on Computer Systems 1,3 (1983), 195-221

/27/ Bode, A.; Fritsch, G.; Händler, W.; Henning, W.; Hofmann, F.; Volkert, J.: Multi-Grid Oriented Computer Architecture. Proc. 1985 Int. Conf. on Parallel Processing, St. Charles 1985, 81-95. IEEE Comp. Soc. 1985.

/28/ Regenspurg, G.: Entwicklung von Zentralprozessoren aus Einheitsbausteinen. Elektron. Rechenanlagen 21 (1979), 61-64, 125-129

/29/ Händler, W.: Thesen und Anmerkungen zur künftigen Rechnerentwicklung. In: Gert Regenspurg (ed.): GMD-Rechnerstruktur-Workshop, München 1980, 17-47

/30/ Bode, A.; Fritsch, G.; Händler, W.; Henning, W.; Volkert, J.: A highly parallel architecture based on a distributed shared memory. Proc. Highly Parallel Computers for Num. and Signal Processing Appl. IFIP Work. Conf. WG 10.3, University of Nice/France, March 24-26,1986

/31/ Linster, C.-U.: SYMPOS/UNIX - Ein Betriebssystem für homogene Polyprozessorsysteme. Arbeitsberichte des IMMD, Bd. 14, Nr. 3, Erlangen 1981

/32/ Bolch, G.; Hofmann, F.; Hoppe, B.; Kolb, H.J.; Linster, C.-U; Polzer, R.; Schüßler, H.W.; Wackersreuther, G.; Wurm, F.X.: A Multiprocessor System for Simulating Data Transmission Systems (MUPSI). Microprocessing and Microprogramming 12 (1983), 257-277

/33/ Bode, A.: Ein Mehrgitter-Gleitkomma-Zusatz für den Knotenprozessor eines Multiprozessors. In: U. Trottenberg, Wypior (eds).: Rechnerarchitekturen für die numerische Simulation auf der Basis superschneller Lösungsverfahren I: GMD Studien Nr. 88, 153-60 (1984)

/34/ Henning, W.; Volkert, J.: Programming EGPA-Systems. Proc. 5th Int. Conf. on Distributed Computing Systems, Denver, May 13-17, 1985, 552-559 IEEE Computer Society Press 1985

/35/ Händler, W.: Fünfte Computer Generation und zukünftige Rechnerstrukturen. To be published: Conference Report of "Die Zukunft der Informationssysteme", Linz/Austria, Sept. 16-18, 1986

PYRAMIDS - EXPECTED PERFORMANCE

M. J. B. Duff

Department of Physics and Astronomy
University College London
Gower Street, London WC1E 6BT

1. INTRODUCTION

When any new computer architecture is proposed, questions are inevitably asked as to how systems based on the architecture can be expected to perform. It is not an unreasonable assumption that each proposal will represent an attempt to produce an improved optimisation against one or more criteria; 'performance', at best a vague, imprecisely defined term, is a measure of how successful that optimisation has been. In this paper, various classes of design criteria will be discussed, particularly in relation to pyramids applied to the processing of image data.

2. DESIGN CRITERIA

It goes without saying that the role of all computer architectures is to enable computer programs to be executed as rapidly as possible within the confines of constraints which will vary from one system to another. These constraints can be regarded as implementing the design criteria. They can be broadly classified as follows:

a) cost constraints, under which the overall monetary cost of the system is to be minimised or, at least, contained within a certain budget;

b) physical constraints, under which factors such as power consumption, size, weight and so on, are to be contained within certain limits;

NATO ASI Series, Vol. F 25
Pyramidal Systems for Computer Vision
Edited by V. Cantoni and S. Levialdi
© Springer-Verlag Berlin Heidelberg 1986

c) algorithmic constraints, under which the system is required to perform optimally in executing a defined set of algorithms, probably at the expense of a degradation of performance elsewhere;

d) language constraints, under which the system is required to perform optimally in executing programs written in a certain language or style of languages.

Cost and physical constraints, although important, will not be considered in this paper. Both are highly dependent on technological developments and on commercial factors and these can change significantly over a short time scale. The discussion will therefore be confined to the longer term factors which are more relevant in an extended programme of research. However, it is important to realise that changing cost structures, such as the decreasing cost of memory in comparison with processors, can strongly influence the way in which resources are utilised in any practical realisation of an architecture. Even so, it does seem to be best to design first with function in mind and only then to see how cost and physical constraints lead to design modifications.

3. ALGORITHMIC CONSTRAINTS

The measure of success which has been enjoyed, mainly in the last decade, by computers specially designed for image processing, has been largely due to the deliberate correspondence between the structure of the computer and the structure of key features of algorithms commonly executed by the computer. This has been most marked in relation to local neighbourhood operations in which a new value for each pixel is calculated as a function of the previous values of the subset of pixels forming a local neighbourhood to the pixel being revalued. The term 'local neighbourhood' can be taken to imply pixels in the image which are near to the pixel under consideration and will often be the 3 x 3 group adjacent to

and centred upon that pixel. Three aspects of these operations are important with respect to optimisation:

a) the operation can, in principle, be performed simultaneously on every pixel in the image, thus allowing massive (fine grained) parallelism to be introduced with, in the limit, one processor for every pixel;

b) since data is required from every pixel in the local neighbourhood, a further degree of parallelism can be introduced in the form of a 'parallel fetch';

c) it is natural to maintain the pixel array structure before, during and after the operation, thus avoiding the heavy overheads inherent in edge following, raster scanning, and so on.

To a greater or lesser degree, these features are to be found in many recent systems including pipelines, conventional mesh-connected arrays and, more recently, systolic arrays. However, the strengths of these systems can also be the sources of their weaknesses. For example, a close match between a two-dimensional data structure and a two-dimensional array of processors will clearly be disadvantageous if the image data subsequently is reduced to an attribute list. Again, limiting interprocessor connections to local neighbourhoods can be very unhelpful when processes such as remapping over long distances are to be carried out.

It is not difficult to state what are the broad requirements for an efficient processor, optimised with respect to a particular algorithm. In the first place, it will provide separate processors for each segment of the operation, the operation being decomposed either spatially and/or temporally. Spatial decomposition implies simultaneous processing of different parts of the image (the parts being even as small as individual pixels) whereas temporal decomposition involves sequential processing of the total image data through a pipeline of processors. An alternative to the latter would be a decomposition of the algorithm into parts which could be

performed at the same time on the same original image data (measuring the integrated optical density of an image in one processor whilst another is extracting edges, for example).

However the decomposition is made, the following aims should direct the course of the system design:

a) a processor should be assigned to each segment resulting from the decomposition;

b) each processor should communicate directly with the sources of the data needed for the task it is to perform;

c) each processor should communicate directly with the destination for the results of its operation;

d) the task decomposition and processor capacity should always be such that each processor is kept usefully busy for the majority of the time.

Individual processors, it will be assumed, will be as powerful as is necessary, or as can be afforded, and will be provided with adequate local memory. Whether or not individual (MIMD) control will also be provided will depend on the task decomposition, but it will probably always be necessary to ensure good communication between each processor and central facilities (data I/O and control). Unfortunately, optimisation for a single algorithm must necessarily imply non-optimisation for others. In practice, therefore, the optimisation will usually be directed to a group or class of algorithms rather than to an individual algorithm so as to provide good performance over a useful selection of tasks.

4. LANGUAGE CONSTRAINTS

A more recent development in computer architectural design practice has been the growing realisation that the value of a computer depends on the type of software that it supports as well as on the efficiency with which it implements algorithms. This became very apparent when attempts were made

to execute programs written in functional languages. Falling costs for hardware components and increasing complexity of programs have combined to change completely the balance between investment in hardware and software. It has also been realised that software validation may be almost impossible with programs written in traditional languages whereas programs written in languages which offer the possibility of formal proof usually run disastrously slowly on conventional hardware.

These facts have combined to stimulate attempts to configure computer hardware so that the structural features of a programming language relate naturally to the computer structure. Experience in this area is not yet sufficiently advanced to prove whether the philosophy is of practical value. Architectures are emerging which relate well to functional programming, data-flow programming or concurrent programming; what is not clear is whether, particularly in image processing, the marked effect that these styles of program have on algorithm design will be advantageous or counterproductive in terms of overall performance. It may be argued that the ease with which the software can be written and validated might offset the cost of a fall in performance, but, if this is the case, the trade-off might not always be acceptable.

5. PYRAMIDS

A pyramid architecture appears to offer certain advantages over, for example, a conventional, 'flat' two-dimensional array. Conceptually, the idea is attractive since it would seem to suggest a way in which the gradual reduction of amounts of data with increasing abstraction, during the process of image analysis, might be matched to a computer structure. However, it should be pointed out that pyramids currently being built only loosely fit in with this philosophy. At a fairly basic and practical level, the main

properties of pyramids in relation to their expected performance can be stated as follows:

a) each level of a pyramid should have all the advantages (and disadvantages) of a conventional flat array;

b) it would not be too difficult or expensive to arrange for each level to be independently controlled;

c) inter-level connections naturally support resolution reduction so that the pyramid is ideally suited to multi-resolution processing;

d) there is a short path between any pair of pixels in the lowest level, being only 2 log N steps for a pyramid with N x N processors in its lowest level and a quartering of the number of processors in successive levels;

e) a pyramid (of the type described above) is not much dearer to build than a flat array of the same base dimensions, requiring approximately one third more processors.

Each of these points will now be examined in some detail.

5.1 Comparison with flat arrays

In principle, each level of a pyramid is a flat array so it might be deduced that each level must in itself exhibit the properties of a flat array. This would be so except for one important detail: each processor in a pyramid has to make connections with (typically) five extra processors, one in the level above and four in the level below. This extra connectivity (and any associated switching) increases the pin-out from the processor(s) chip and also reduces the silicon area available for processing and/or on-chip memory. If these factors can be overcome, then the single layer performance throughout the pyramid need not be significantly degraded and the first property listed above will be valid. From the point of view of system cost, it must be expected that less

processors could be packed into a chip (compared with a flat array) and performance per unit price must be reduced.

5.2 Independent levels

A pyramid could be treated as a pipeline of flat arrays with automatic resolution reduction between the stages. The cost of introducing a separate controller for each level of the pyramid would not be excessive, (1 + log N) controllers being required. Seen in another way, for a large pyramid (N > 64) with (N x N) processors in its lowest level, the cost of adding a further lower level would be approximately three times the cost of the original pyramid and would completely dwarf the cost of supplying the extra level with its own controller.

The provision of independent control for each level would permit, in principle, different types of processing to be carried out simultaneously at each level and would therefore offer a maximum speed gain of (1 + log N), although great care would have to be taken in order to synchronise the transfer of results between the levels. In practice, it is doubtful whether the gain would be much more than a factor of two or three, because of the synchronisation problem, bearing in mind that even the largest pyramids proposed so far have less than ten levels. Assuming that new images are input to the lowest level, some useful gains would be achieved by continuing with the processing in the second or higher levels whilst the lowest level is 'hung up' by the inputting. The same effect could be achieved by buffering in the lowest level, although this would involve some extra circuitry. Generally speaking, the additional registers and gates would involve an almost negligible increase in the processing element complexity.

5.3 Multi-resolution processing

It is in this area that pyramids show their greatest strength and it is here that comparisons with flat arrays can most profitably be made. Typically, resolution reduction involves forming averages of 2 x 2 pixel sub-areas of the image. In a flat array with 4-connectivity, this can be accomplished by one parallel shift in the x direction, an addition, a parallel shift in the y direction and a further addition. However, the valid pixels are then distributed across the array as shown in Fig. 1 and need to be compacted into a contiguous array in order to re-establish local neighbour connectivity. This can be achieved by masking out the invalid data and shifting all but the leftmost column of pixels one place to the left, repeating the process for all but the two leftmost columns, and so on until all the pixels are to the left of the array. The process is then repeated on rows, moving these up until the valid pixels are in the top left-hand corner of the array. The whole process requires 2N shifts and 2N masking operations, in addition to the two shifts and adds at the start. In an 8-connected array, 2N shifts and masking operations would suffice.

V * V * V * V
* * * * * * *
V * V * V * V
* * * * * * *
V * V * V * V
* * * * * * *

Fig. 1 Valid pixels are shown as V and invalid pixels as *.

It is clear that this operation would be slow in an array with conventional nearest neighbour connectivity, but it is useful to consider whether a simple extra connectivity would

provide the necessary enhanced speed. As an example, the following proposal is worth studying. For simplicity, the one-dimensional case is shown in Fig. 2.

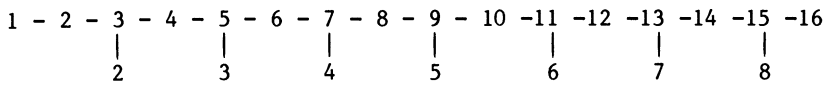


Fig. 2 Extra connectivities for multi-resolution processing.

In general, and in two dimensions, a processor with (x,y) coordinates in the array (2p-1,2q-1) must have an outgoing connection to the processor in the array position (p,q), for all integral values of p and q. With this connectivity, one parallel shift along the extra connections, first in the x direction, say, and then in the y direction, would compact the data into the smallest possible top left-hand corner of the array. It therefore seems that the addition of only one extra input connection to each of one quarter of the processors would achieve the pyramid-like data compaction needed for multi-resolution processing. Note that the same connections provide the compaction equivalent for all the pyramid levels. Fig. 3 shows the connection pattern for an 8 x 8 array in which the processor with an upper case label outputs to the processor with the corresponding lower case label.

It will be noted that connections in this configuration eliminate the need for masking, although it is important to realise that data will be valid only in the $N/2^C$ x $N/2^C$ array in the top left sector of the N x N array, where C is the number of compaction shifts which have been executed (i.e. the valid data will be in an array of size equal to the appropriate level of the equivalent pyramid). It is therefore clear that the multi-resolution function of the pyramid can be efficiently emulated on a flat array provided one connection is added to each of a quarter of the N x N processors.

a	b	c/B	d	C	*	D	*
e	f	g	h	*	*	*	*
i/E	j	k/F	l	G	*	H	*
m	n	o	p	*	*	*	*
I	*	J	*	K	*	L	*
*	*	*	*	*	*	*	*
M	*	N	*	O	*	P	*
*	*	*	*	*	*	*	*

Fig. 3 Compaction in an 8 x 8 array.

5.4 Short paths

The second important characteristic of pyramids is the maximum path of 2 log N steps separating any pair of processors in the lowest level. Careful examination of this claim soon reveals a problem when blocks of data are to be moved to new array positions: the bandwidth for transfer of pixels decreases by a factor 4 at each higher level of the pyramid. In the arguments that follow, certain simplifying (but optimistic) assumptions will be made:

1. A processor can execute two functions during one instruction cycle:
   a) transfer four pixels from the level below it and store them in its local memory,
   b) transfer one pixel from its local memory to the level above it.
2. Data can move simultaneously between all levels of the pyramid, but only in one direction at a time, either ascending or descending.
3. There is no penalty for routing the pixels in the required directions during the descent from the top of the pyramid.

4. There is no penalty for local memory address calculations during either the ascending or descending parts of the transfer.
5. All the descending transfers can be effected at the same rate as the corresponding ascending transfers.

Consider first the extreme case in which every pixel in the base level is to be moved to a new position in the same level. By carefully programmed routing, it should often be possible to achieve this without forcing every pixel through the highest processor. Nevertheless, in the worst case, all the pixels from the base level would need to be raised to the top of the pyramid where, it can be assumed (in the absence of storage constraints), they would be accumulated before any start the downward journey to their final locations. Assuming (N x N) to be very large compared with log N, the time required for this process will be very close to the time needed to transfer (N x N) pixels through the four connections up to the top level processor, i.e. (N x N)/4 cycles. Similarly, in the descending sequence, accepting the fifth assumption above as valid, during each cycle four pixels will be extracted from memory in the highest processor and distributed into the memories of processors in the level below. This repeated operation again dominates the process and takes the same number of cycles as the ascending process. Thus the complete transfer would take (N x N)/2 cycles. Note that if pixels were to be transferred independently, one at a time, from the base level, through the top of the pyramid, and back to the base, the number of cycles needed to transfer all the pixels would be 2(N x N) log N.

A second example worth consideration is the translation of a complete row of pixels to a new position in the base level. In this case, N pixels have to be transferred through only two connections to the top processor. During the ascent, the first pixels would reach the level below the top after (log N - 1) cycles and a further N/2 cycles would be needed to store all the pixels in the top processor's memory.

The descending sequence would take an equal number of cycles so that the complete process would require (N-2 + 2 log N) cycles.

In order to evaluate these estimates, it is instructive to compare them with results to be expected from flat arrays. These arrays will often have a rapid broadcast facility by which the contents of a memory element in a selected register can be sent in parallel to all other processors in one instruction cycle. Using a masking field or activity bit, the broadcast value is then stored only in the selected 'destination' processor memory. If each pixel value is represented by B binary bits, all the pixels can be moved in approximately 3B(N x N) cycles. Alternatively, if each pixel has its destination address stored with it and, if the local address is subtracted from it (forming a relative address), then the transfers can be effected by a series of parallel shifts after each of which the relative addresses are decremented. Each shift will comprise a masking operation (to determine which pixels are to be shifted), a shift for each of the B bit planes representing the pixel values and a shift for each of the 2 log N bit planes representing the relative addresses, plus 2 log N decrements (these being bit-serial operations in most arrays). An average number of N parallel shifts will be required (N/2 in both x and y directions), implying approximately N(4 log N + B) cycles. For 8-bit pixels and N larger than 64, this further approximates to 4N log N cycles, which is significantly smaller than 3B(N x N). In real applications, it is usually found that distances to be moved are much less than N and that, consequently, the relative addresses are smaller.

The comparisons between pyramids and flat arrays for transferring all pixels are summarised below:

	Parallel method	Individual transfers
Pyramid	(N x N)/2	2(N x N) log N
Flat array	N(4 log N + B)	3B(N x N)

These estimates clearly do not imply a significant advantage in using a pyramid.

The comparison for translation of a row of pixels is more striking since the parallel shift method in a flat array would require at the most BN cycles or, if it is considered reasonable to assume that connections in a flat array are B bits wide (as already assumed for pyramids), N cycles. This compares with (N-2 + 2 log N) cycles for pyramids. Finally, using either propagation or rapid broadcast, the flat array estimate can be reduced at least by an order of magnitude. As an example, in CLIP4 the time taken to move a row of B-bit pixels across the full array is only (2 + N/40)B cycles, since each propagation step takes 0.2 μs in addition to the 8 μs for an instruction cycle.

In summary, it should be noted with caution that the apparently unchallenged superiority of the pyramid in conveying data across large arrays is not always justified in practice. To be pessimistic, unless the communication channels in the pyramid are particularly well engineered, it is likely that better transfer rates might be achieved for transfers confined within the base array.

5.5 Low cost

The cost of converting a flat array into a pyramid is usually stated as being approximately an additional one third, being the fractional increase in the number of processors. Since this number of processors would only increase N in a flat array by about 15%, converting a 64 x 64 array into a 74 x 74 array, for example, the interesting possibilities of the pyramid structure far outweigh the value of the marginally increased resolution obtained by adopting the less enterprising way of employing the extra processors. In practice, the additional between-level connections also increase the cost but, in a research environment, this might be an insignificant factor. On the whole, it can be sensibly decided that the pyramid structure is not likely to make unacceptable financial demands in comparison with the equivalent size flat array.

6. HIGH-LEVEL AND LOW-LEVEL PROCESSING

Pipeline processors, systolic arrays and the now-conventional flat, SIMD arrays, are probably adequately efficient when used for low-level processing but are sadly lacking when implementing the less regularly structured high-level algorithms. In a complete image analysis program, the bottleneck is now almost always to be found in the later parts where the high-level operations are being performed.

If one can accept this as a true statement, then it is prudent to ask (a) whether there is a need for another structure to execute the low-level algorithms and (b) whether pyramids have anything additional to offer at the high-level end of the processing?

It is the reluctant conclusion of the arguments presented in this paper that the answer to the first question must be no. Pyramids do not seem to offer substantial advantages over flat arrays for low-level operations. As to the second question, it may be too early to attempt an answer. The rigid connection structure of the pyramid does not encourage one to think that it will be readily adaptable to the widely varying data formats resulting from the high-level processes. It also can be argued that the processors at the higher levels of the pyramids should be more powerful as the data abstracted from the image becomes more complex. Nevertheless, whenever a new computing structure emerges, it takes time for users to discover how best to shape algorithms which will take advantage of the unusual properties of the structure. The pyramid's potential has still to be explored.

7. BIBLIOGRAPHY

This paper has tried to explore some of the claims made in the literature for the expected performance of pyramid computers. Assessment of performance is always a difficult exercise and no clear rules of procedure have been established when the performance of image processors is under

consideration. References on the proposed design and use of pyramids are listed elsewhere in this volume and detailed discussions on performance estimation for a range of image processing architectures are to be found in "Evaluation of Multicomputers for Image Processing" (eds. Uhr, Preston, Levialdi and Duff), Academic Press, New York, 1986.

HYPERCUBES AND PYRAMIDS

Quentin F. Stout
Electrical Engineering and Computer Science
University of Michigan
Ann Arbor, MI 48109 USA

1. INTRODUCTION

Hypercube computers have recently become popular parallel computers for a variety of engineering and scientific computations. However, despite the fact that the characteristics which make them useful scientific processors also makes them efficient image processors, they have not yet been extensively used as image processing machines. This is partially due to the hardware characteristics of current hypercube computers, partially to the particular history of the groups which first built hypercubes, and partially to the fact that the image processing community did not initially realize some of the advantages of hypercubes. In this paper, their suitability for image processing will be put forth, showing that they can be viewed as competitors to, and collaborators with, mesh and pyramid computers, architectures which are often promoted as being ideal for image processing.

First some of the general graph-theoretic properties of hypercubes will be given. Second, the primary reasons for the initial interest of the engineering and scientific community, namely hypercubes' suitability for mesh calculations and as general purpose parallel computers, will be shown. Third, the ability of hypercubes to efficiently execute pyramid algorithms is shown, an ability which has not yet been taken advantage of. Fourth, currently available commercial hypercubes are examined to show some of the trends in the increasing sophistication of their implementation. Finally, a new architecture is suggested which combines hypercubes and pyramids to make machines capable of rapidly processing large images and performing image analyses from lower level image processing to higher level image understanding.

Pyramid computer means the standard model used in most of the other papers in this volume, namely, a generic processing element (PE) is connected to four neighboring PEs on the same level, a parent PE on the level above, and four children PEs on the level below. **Mesh computer** will usually mean a square 2-dimensional mesh in which a generic PE is connected to four neighbors, but in some clearly understood cases it will mean a mesh of any dimension. Hypercubes will be defined below. For positive func-

NATO ASI Series, Vol. F 25
Pyramidal Systems for Computer Vision
Edited by V. Cantoni and S. Levialdi
© Springer-Verlag Berlin Heidelberg 1986

tions f and g defined on the positive integers, the notation $f = \Omega(g)$ ("f is of order at least g") means that there are constants $C, N > 0$ such that $f(n) \geq C*g(n)$ for all $n \geq N$, and $f = \Theta(g)$ ("f is of order exactly g") means that there are constants $C, D, N > 0$ such that $D*g(n) \geq f(n) \geq C*g(n)$ for all $n \geq N$. For example, $n*\sin(n)^2 + \log_2(n) = \Omega(\log_2(n))$, and $3n + n^2 = \Theta(n^2)$.

2. HYPERCUBES AS GRAPHS

Hypercubes have long been studied by graph theorists. For integer dimension $d \geq 0$, the (binary) **hypercube of dimension *d***, denoted H_d, is as a graph of 2^d vertices with unique d-bit binary strings used as labels, where there is an edge connecting two vertices if and only their labels differ by a single bit. Often the labels will be interpreted as binary numbers. Figure 1 shows hypercubes of small dimensions. A hypercube computer is formed by placing a PE at each node, and a communication link along each edge. This makes a local, or distributed, memory machine where information is passed as messages between PEs, as opposed to a global, or shared, memory machine where information is exchanged by placing it in the global memory.

Several properties of hypercube graphs are immediately apparent. First, they are **homogeneous**, meaning that for any dimension d, given any two vertices p,q in H_d, there is a graph isomorphism of H_d onto itself which maps p onto q. To see this, let r = label(p) xor label(q) (all logical operations are performed bitwise). The mapping which maps a vertex s to the vertex labeled r xor label(s) is one such isomorphism. Homogeneity implies that all nodes can be treated equally, and in particular means that in a computer implementation it is natural to allow input/output to all nodes. It also means that if an algorithm treats a node specially (for example, if node 0 is used as the root of a tree), then by using xor the algorithm can be "translated" so that any other desired node is the special one. Pyramids and meshes are not homogenous, since the apex is unique and corners can only be mapped to other corners, but tori are homogeneous. Binary hypercubes are special types of tori.

Routing messages between nodes is particularly simple in a hypercube. A message from node p to node q must travel along at least as many edges as there are 1s in the xor of p and q's labels, and there are paths which attain this lower bound. Further, such paths are quite easy to compute dynamically: if a message destined for q is currently at p, then let i be the location of any 1 the xor of p and q's labels, and have p forward the message to the neighbor with a label differing in the i^{th} bit. Notice that there are many such paths of minimal length, which can allow for routing variations that do not increase the path length. The **diameter** of H_d, i.e., the largest distance between nodes, is d, or $\log_2$(number of nodes). In a 2-dimensional mesh the diameter is the square root of the

number of nodes, while in a pyramid it is $2*\log_2$(number of nodes in the base).

Each node in H_d has **degree** d, meaning that it has d edges. In a physical implementation the degree of some nodes must be d+1 to allow communication to the outside world, so if communication is homogeneously implemented then all nodes will have degree d+1. To build hypercubes of increasing size, the degree must also increase, which makes the individual nodes slightly more complex. Meshes and pyramids of fixed dimension but increasing size do not have this problem, which at some point can become a limiting factor on the size of hypercubes. As will be shown below, current technology makes it possible to build hypercubes with thousands of nodes, so the node degree is not as serious a problem as it was in the 1970's, during which time alternatives such as cube-connected cycles [6] were suggested to alleviate this problem.

Hypercubes are eminently partitionable into smaller hypercubes. For example, H_{d+1} can be partitioned into two disjoint copies of H_d by taking one of the d+1 coordinates (bit positions in the label) and using all nodes with a 0 in that coordinate as one copy of H_d, and all nodes with a 1 in that coordinate as the other copy. More generally, it is easy to show that if one is given a collection of hypercubes and wants to embed them in non-overlapping fashion in a given hypercube H, then this can be done if and only if the total number of nodes in the collection is no more than the number of nodes in H. (Embedding means that no two nodes are mapped to the same node, and that neighbors are mapped to neighbors.) In a multiuser environment, this means that it is quite easy to allocate subcubes to different users, and there is a great deal of flexibility possible for dynamic allocation.

3. MESH AND GENERAL PURPOSE CALCULATIONS

Many scientific and engineering applications deal with data organized as a matrix where the operations performed on an entry involve nearby elements of the matrix. For example, relaxation methods for solving partial differential equations update a point based on its current value and the value of its nearest neighbors. One quite useful property of hypercubes is that meshes (matrices) of all dimensions can be embedded in them, as shown below, so a matrix can be distributed in a hypercube such that each PE can update the values of the entries it contains by using only its own entries and the entries of neighboring PEs. This miminizes communication time, and is as good as any mesh computer with the same number and type of PEs. In other problems the communication between PEs is quite irregular, in which case the hypercube is useful because it is a fairly fast general purpose message delivery system. Thus the hypercube supports problems with both regular and irregular communication patterns.

3.1 MESHES

Meshes are embedded into hypercubes by using Gray codes. Given positive integers i and k, a k-bit **Gray code** G for $0...i$ is a 1-1 mapping from $0...i$ into k-bit binary strings, where $G(j)$ and $G(j+1)$ differ by exactly 1 bit for all $0 \le j < i$. Since the mapping is 1-1, it must be that $k \ge \log_2(i+1)$. A particularly useful collection of Gray codes are the reflexive ones G_d. For all positive integers d, G_d is an d-bit Gray code for $0...2^d-1$, recursively given by $G_1(0) = 0$, $G_1(1) = 1$, and $G_{d+1}(j) = 0G_d(j)$ (that is, the concatenation of 0 and $G_d(j)$) for $j<2^d$, and $G_{d+1}(j) = 1G_d(2^{d+1}-1-j)$ for $2^d \le j < 2^{d+1}$. Figure 2 shows these reflexive Gray codes for small sets. From now on only these Gray codes will be used, and their special properties, beyond those of generic Gray codes, will be used in the pyramid mappings.

A matrix $a[\ 0..2^{i1}-1, 0..2^{i2}-1, \ldots, 0..2^{id}-1\ ]$ can be mapped in a 1-1 fashion onto a hypercube of dimension $i1+i2+...+id$ by mapping $a[j_1, j_2, ..., j_d]$ to the node labeled $G_{i1}(j_1)\ G_{i2}(j_2)\ ...G_{id}(j_d)$. Adjacent matrix entries differ by exactly 1 in one coordinate, so they are mapped to nodes with labels that differ by exactly 1 bit, that is, they are mapped to adjacent nodes. (Besides being easy to compute, the reflexive Gray codes have the additional property that they also preserve toroidal neighbors, where neighbors are computed in a modular fashion.) Therefore matrix operations which update an entry based on entries in a local neighborhood can be implemented so that a node need only obtain entries for nearby nodes. This has been exploited in hypercube implementations of scientific applications, and of course it applies equally well to local operations on images. Figure 3 shows a mapping of a 2-dimensional mesh onto a hypercube. Such a mapping also maps nonadjacent matrix positions to adjacent nodes, much like crumpling a piece of paper maps points far apart on the 2-dimensional paper surface to nearby points in 3-dimensional space.

If the matrix has more entries than there are nodes in the hypercube, then the above mapping must be modified. The matrix should be partitioned into equal sized rectangular blocks, where the number of blocks equals the number of nodes. The blocks themselves form a d-dimensional mesh, so they are mapped to the nodes as before. (This assumes that a node can hold an entire block.) If the blocks are of equal size then each PE will have roughly the same amount of work to do, so PEs will not sit idle waiting for others to finish. (It is only roughly the same amount of work because the calculations may not be identical at all entries, and in particular edge entries may need to be computed differently.)

Another important consideration is the ability to overlap calculations with PE/PE communication. While this would not be very important if such communication were as fast as individual calculations, in practice communication is significantly slower. This

can become the dominate factor in determining the running time, and can significantly decrease the efficiency of the parallelism. The **efficiency** of an n PE parallel machine running a program is defined to be $T_1/(n*T_n)$, where T_n is the time of the program and T_1 is the time of the fastest program solving the same problem on a machine with a single PE. The time for PE/PE communication does not appear in T_1 , but does in T_n , which is why it decreases efficiency.

For near-neighbor operations on a matrix, overlapping computation and PE/PE communication is fairly straightforward and is commonly used whenever the hardware can support it. For example, in a 2-dimensional matrix, suppose each entry gets a new value which is a function of its current value and the values of its four nearest neighbors. (We will ignore what happens along edges of the matrix.) In each block of entries assigned to a single PE, entries along the edges need to be updated using the values of entries from neighboring PEs, and conversely, neigbhboring PEs need the values of these entries. However, entries in the center can be updated using only information currently in the PE. To overlap operations, each PE sends the values of entries on the right edge of its block to the neighboring PE holding the block to the right, sends the top edge to the PE holding the block to the top, the left edge to the left PE, and the bottom edge to the bottom PE. It is assumed that these transmissions can be performed by initiating them and then proceeding to other calculations while the transmissions are completed. While they are being completed, entries in the center are updated. If the transmissions are completed while the center entries are updated, then the edge entries can be updated without having to wait for neighboring values to arrive. The time to initiate the communications will still appear in T_n and not in T_1, but this should be much less than the time to complete communication.

3.2 GENERAL MESSAGE TRANSMISSION

One common global operation, **broadcasting**, sends data from one PE to all other PEs, and another common operation, **reporting**, sends and condenses information from all PEs to a single one. In reporting the condensation may be to sum values, taking an or of logical values, taking the maximum, etc. Even with matrix input, these global operations may be needed to perform tasks such as determing average gray level and sending it to all PEs so they can use it to set a threshhold.

Broadcasting can be quickly performed on a hypercube using **recursive doubling**. If PE 0 has the data to be broadcasted, then at time 1 it sends the data to PE 1. (Time units are taken to be the time required to have a message sent and received, and to determine the next PE to send to.) At time 2 PE 0 sends the data to PE 2 and PE 1 sends the data to PE 3. In general, at the end of time i PEs $0 \ldots 2^i-1$ have the data, and during time unit $i+1$ each one of them sends a copy to the neighbor 2^i greater. This continues until all

PEs are informed, taking d time units for H_d. If the condensation occuring in reporting is a semigroup operation such as summing or finding a maximum, then recursive doubling can be performed in reverse (recursive halving), where each PE receiving a value combines it with its own, taking d time units to complete a report. Here the time unit includes the time to do the condensation. For both operations, since the diameter is d this is the best one could hope for, and it can be achieved even if each PE can communicate with only one other at a time. Further, by using the homogeneity properties noted above, any PE can play the role of the distinguished PE, without any loss of time (except for the cost of performing xor). In a pyramid, broadcasting or reporting is best if the apex is used as the distinguished node. If each PE can communicate with all four of its children in one time unit, then broadcasting and reporting can be done in $\log_2(n)/2$ time units, where n is the number of PEs at the base. With one-at-a-time communication and the simplest approach, the time is $2*\log_2(n)$. (Less natural approaches can reduce this.)

A far more complicated activity is to rapidly transmit messages from many PEs to "random" or "bad" destinations. For example, by using a wire-cutting argument it is easy to show that if a pyramid has a base of n PEs, and if each PE sends a message to a destination given by a uniformly generated random permutation, then the expected time to deliver all messages is $\Omega(n^{1/2})$. This occurs despite the fact that the diameter of a pyramid is $\Theta(\log_2(n))$, and in fact for this type of computation the pyramid is no better than a 2-dimensional mesh. For H_d, however, by using Bitonic sort where the message address is used as the key, any possible permutation of messages can be delivered in $\Theta(d^2)$ time, i.e., in $\Theta(\log^2(n))$ time, where n is the number of PEs. By using randomization, Valiant showed that any message permutation could be delivered in $\Theta(d)$ expected time [12], although it is not clear whether the greater complexity of this algorithm can ever beat bitonic sort on the sizes of hypercubes to be available in the foreseeable future. Further, empirically it seems that the simple strategy used on all current MIMD hypercubes works exceedingly well. This strategy is to have each PE keep a small queue of messages to be forwarded, and at each time step take the next message and forward it by using the routing algorithm mentioned in Section 2. Either a high-to-low or low-to-high scan is used to find the next 1 in the xor'ed pattern showing which coordinates need to be traversed.

4. EMBEDDING PYRAMIDS IN HYPERCUBES

The material in the previous sections has been widely used in solving various scientific and engineering problems on hypercubes. In this section we show the less well known fact that hypercubes can also efficiently execute pyramid algorithms by simulating pyramids. Unlike meshes, pyramids cannot be embedded in hypercubes so that neighbors in the pyramid are mapped to neighbors in the hypercube. This is because a

cycle proceeding from a parent to one child, then to an adjacent child, and back to the parent, has an odd number of nodes, while all cycles in a hypercube have an even number of nodes. Instead, one can ask that the pyramid be mapped into a hypercube so that neighbors in the pyramid are mapped to PEs close to each other in the hypercube.

As with meshes, the most important case is where the number of PEs in the hypercube is less than the number of pixels, i.e., less than the number of PEs in the base of the pyramid. For example, the hypercube with the largest number of PEs presently available, the Connection Machine [2], has 64K PEs, while a 512×512 image has 256K pixels, and a 1024×1024 image has 1024K pixels. As before, the best way to map the base onto the hypercube is to subdivide the base into squares, where the number of squares equals the number of PEs. Each PE will simulate its base level pyramid nodes, as well as all nodes in higher levels which sit only above the nodes in the square. For example, using H_{10} on a 512×512 image, each PE will receive a 16×16 square, so each PE will simulate the work of a subpyramid with a 16×16 base, a total of 5 levels. Just as for the base, for each of these levels, each pyramid node will either have all of its neighboring pyramid nodes at the same level in the same PE or in neighboring PEs. For pyramid nodes in these levels, the only parent/child pairs perhaps not in the same PE are the apex of the subpyramid and its parent.

For the next level up, notice that these nodes are parents of the apexes of subpyramids in 4 PEs, where the 4 PEs form a square. One of these PEs will also do the work of simulating the node at the next level. Therefore one of the PEs must send its data over two communication links to reach the parent. To pick which PE to use, notice that the reflexive Gray codes have the property that in each square exactly one of the rows and one of the columns has a least significant bit of 0 in its Gray code. If the PE in this row and column is used to simulate the parent, then it is easy to see that pyramid neighbors in this higher level are again simulated by neighboring PEs. Figure 4 shows the chosen nodes in each square for a small example. The PEs simulating this level form a subcube of dimension 2 less than the original one, so to simulate pyramid nodes at still higher levels the process is repeated within this subcube. When all nodes are simulated in this manner, no pyramid neighbors are separated by more than two communication links.

When H_{10} simulates a pyramid with a 512×512 base (a pyramid with 9 levels), PE 0 simulates nodes on all 9 levels. It simulates a total of 346 nodes, while some other PEs, such as PE 1, simulate only 341 nodes. If the pyramid algorithm is such that the amount of work is the same at each node, then PE 0 has to do only about 1% more work than any other node, so the load is well balanced.

Even if the number of hypercube PEs exceeds the number of pyramid nodes, the time used by the above scheme is approximately as good as possible if the pyramid algor-

ithm proceeds level by level , for then no PE need simulate more than one node at a time. However, a PE simulating nodes on several levels may spend some extra time in switching from one simulation to the next, and may need extra space to store local variables of the simulated nodes. If the algorithm uses all levels simultaneously then the above scheme will have one PE simulating a number of pyramid nodes equal to the pyramid's height plus 1, which means that it can take many time units to simulate a single step of the pyramid. In this situation one can do slightly better. If the dimension of the hypercube is just large enough to exceed the number of pyramid nodes, then the hypercube has exactly twice as many PEs as there are nodes in the pyramid's base. The base should be simulated using the subcube which is the lower numbered half of the hypercube. Using the same representatives of squares as shown in Figure 4, now a parent is simulated by the PE in the upper half subcube which is connected to the representative used previously, i.e., the simulation has been moved along a coordinate. This parent is now three communication links from one of its children. If the hypercube had additional unused dimensions the process would just be repeated, moving along a new dimension for each level. Instead, note that this first level of parents does not using any PE in its half which has a 1 in its least significant bit. Therefore the least significant bit can be treated as a "new" dimension. This process of "reusing" dimensions can be continued to represent all levels of the pyramid, with each PE simulating at most one pyramid node, and with pyramid neighbors mapped to PEs at most three communication links apart.

For completeness, we note that when the number of hypercube nodes is at least twice the number of base nodes in the pyramid, it is possible to map the pyramid in a 1-1 fashion into the hypercube so that no two pyramid neighbors are more than two communication links apart. To do this, some neighbors on the same level, as well as some parent/child pairs, must be mapped to PEs two links apart. Suppose the pyramid base (level 0) is $2^b \times 2^b$, so the hypercube is H_{2b+1} . Define $E(i,j)$ to be 0 if $j = 0$, to be 00 if $j > 0$ and $i \equiv 0$ or 3 mod 4 , and 11 if $j > 0$ and $i \equiv 1$ or 2 mod 4 . Define G_d to be the empty string if $d \leq 0$. Then one such mapping is to map the node at level h , row i, column j to PE $G_{b-h}(i)\ (01)^{h/2}\ G_{b-h-1}(j \text{ div } 2)\ E(j, b-h)\ (01)^{h/2}$ if h is even, and to PE $G_{b-h-1}(i \text{ div } 2)\ E(i, b-h)\ (01)^{(h-1)/2}\ G_{b-h}(j)\ (01)^{(h+1)/2}$ if h is odd. Because of space limitations, no explanation of this formula will be given, but it is relatively straightforward to verify that it has the desired properties.

It should be noted that not all pyramid algorithms should be simulated on a hypercube. For problems in which pyramid algorithms do about the same amount of work at each level, usually the pyramid is about the best possible architecture, and other papers in this volume give examples of such algorithms. However, for some problems the best pyramid algorithms must work longer at higher levels of the pyramid [3]. For such problems, sometimes there are hypercube algorithms which utilize all of the hypercube connections and are faster than simulating a pyramid [4].

5. HYPERCUBE MACHINES

Hypercube computers were first proposed by Squire and Palais, at the University of Michigan, in 1962 [8,9]. Their study showed that, at that time, a 12-dimensional hypercube would need about 20 times as many parts as IBM's Stretch, a rather unreasonable component count. Another large paper study was performed by Sullivan et al in 1977 [10,11]. Finally, in 1983, the "Cosmic Cube", a hypercube computer with 64 nodes, was built at the California Institute of Technology [7].

In 1985 Intel Corporation delivered the first production hypercubes, the iPSC, based on the CalTech hypercubes. It has a maximum of 128 nodes, where each node uses a 80286/80287 processor and has 512 Kbytes of memory. Only a single user can use the cube at one time (thought several can use the host at once), and there is only a single I/O channel between the host processor and the hypercube. The time for PE/PE communication is quite high, requiring several milliseconds. A few months later Ametek introduced a very similar hypercube, with the primary changes being a maximum of 256 nodes, and an additional 80186 coprocessor on each board to handle communication. The extra coprocessor allows additional overlapping of communication and processing, and for messages traversing several communication links the intermediate 80286 processors need not be interupted. Because of very slow I/O between the cube and host, neither of these machines is suitable for real-time vision.

In late 1985 NCUBE introduced a hypercube with a maximum of 1024 nodes, each of which uses a proprietary processor [1]. A single chip handles all node I/O via DMA, all standard processing on 32 bit words at about 1 million instructions per second, and floating point arithmetic at about 0.5 MFLOP on single precision reals. Each PE contains only 128 Kbytes of memory. Each PE has its own I/O channel to a host board, providing a maximum system I/O rate of about 720 Mbytes/sec. A multiuser operating system is provided for the hypercube. PE/PE communication proceeds at about 1 Mbyte/sec., with the startup time heavily dependent upon the protocol used. The University of Michigan is one of the beta test sites for this machine, and we hope to implement several vision applications, making use of its high I/O rates as well as high processing rates. The proposed implementation of one application is discussed in [5].

All of the machines mentioned above are MIMD, where each PE has its own program and executes it separately (except, of course, for communication) from other PEs. Most applications use the SCMD, "Single Code, Multiple Data" mode where all PEs are executing the same program, although at any point in time they may be following different branches. A SIMD bit-serial hypercube, the Connection Machine, has been built in 1986 by Thinking Machines Corporation [2]. This is a 12-dimensional hypercube, where each node contains 16 PEs. To process image data there is a special I/O mechan-

ism which connects all the PEs as a 256×256 mesh. Each PE contains only 4K bits of memory. While this machine will be quite useful for some vision and artificial intelligence applications, its SIMD nature forces drastic revisions in algorithms, and the small amount of memory per PE can prove debilitating.

Also in 1986, two commercial hypercubes of vector processors were announced. The Intel iPSC-VX machine is based on the iPSC, with a maximum of 64 nodes. Each node consists of a iPSC node on one board, and a vector processor on a second board, with a total of 1.5 Mbytes of memory per node. All communication and control functions are performed by the iPSC node. The FPS "T Series" machine has a maximum of 16,384 nodes. Each node uses an Inmos Transputer for communication and control, with an additional vector processor, and has 1 Mbyte of memory. Initially the machine must be programmed in Occam, using subroutine calls to utilize the vector processor. Both machines currently allow only one user on the hypercube at a time.Both of these machines have been designed for engineering and scientific computations, with the vector processing being of miniscule utility for image processing.

6. HYPER-PYRAMIDS

One can borrow some of the ideas mentioned in preceeding sections to design a machine capable of rapidly processing large images and performing a range of image processing from lower level operations to higher level ones. The hypercubes of vector processors are an attempt to provide relatively inexpensive supercomputer performance by combining the best vector processors that can be made from inexpensive, commercially available parts, with a hypercube interconnection to provide inexpensive parallelism. The Connection Machine similarly combines the simplicity of mesh computing with hypercube interconnections for general purpose message delivery. The NCUBE machine and Connection Machine emphasize the importance of high I/O rates to balance high computation rates. Further, the hypercube can rapidly simulate meshes of any dimension, slightly less rapidly simulate pyramids (since neighbors are mapped to PEs two links apart), and act as a fairly fast general message delivery facility. Finally, as other papers in this volume show, pyramid algorithms seem to be quite useful for lower levels of image processing. For higher levels they are currently less useful, and a more flexible interconnection structure may be needed. Since the higher levels are poorly understood, it seems premature to optimize a machine to perform any given algorithm, but rather one should build a machine capable of running a variety of algorithms without imposing serious bottlenecks.

This suggests a hypercube of pyramid processors, a **hyper-pyramid**. Each node will have a master processor which is responsible for communication, control, and gen-

eral purpose computing, and a pyramid processor. The pyramid processor need not be a physical pyramid, but must be capable of rapidly executing pyramid algorithms. Other papers in this volume suggest such processors. The master processor, like the NCUBE processor, should be capable of supporting very fast PE/PE communication. Each node should also have a high speed I/O link to the outside world. It may be that I/O is through the master processor, or, as in the Connection Machine, through the pyramid processor. The master processors will form an MIMD hypercube. The pyramid processor may be an SIMD machine, as would likely occur if it was a physical pyramid or mesh, where each master processor controls its own pyramid processor. Or, the pyramid processor may be a raster scan processor, in which case the machine would be very similar to the Intel iPSC-VX and FPS T Series machines.

The size of pyramid that the pyramid processor operates on is primarily dictated by cost and technology tradeoffs. It may be, for example, that the machine is designed to process 512×512 images, using 64 nodes each processing a 64×64 square. To run a 9 level pyramid algorithm over the entire image, the lowest 6 levels would be performed by the pyramid processors, and the top 3 levels would be performed by the master processors. If the work at each node is the same, then more than 99.9% of the work will be done by pyramid processors. On large images, often the top few levels are not used, so it may be that the master processors would rarely be needed in this fashion. However, for higher levels of image processing they will be crucial. For example, they may work together to make some initial global decisions concerning approximate locations of objects, and then each may direct its pyramid to look for specific features to confirm or refute such decisions. Since the master processors would be general purpose processors, they will be easier to program and they permit easier alterations. The pyramid processors may be harder to program, and probably would be used primarily by calling packaged routines. Since lower level image processing is somewhat better understood, these routines need not change as frequently.

One interesting question is whether master processors specially designed for such machines will soon be commercially available. The Transputer is an attempt at such a processor, but it provides only 4 bidirectional communication links which must be multiplexed in order to provide sufficient near-neighbor connections for a hypercube. The NCUBE node processor comes closer to what is desired, but it should be possible to build even better master processors. FPS and Intel have already produced "hypercube of X" machines for X="vector processor", and this paper proposes such machines with X="pyramid processor". There are presumably other interesting choices for X, so there may eventually be enough demand to provide high-quality commercial master processors for such machines.

7. CONCLUSION

We have shown that the hypercube can perform mesh calculations as fast as a mesh computer, in that mesh neighbors are mapped to hypercube neighbors, and it can perform pyramid calculations almost as fast as a pyramid computer. Besides supporting such regular communication requirements, it provides good support for irregular communications between processors, the sort of communication which seems prevalent in artificial intelligence and higher level image processing. Thus the hypercube seems a nearly ideal architecture for image processing and understanding, particularly when the emphasis is on using one machine to do all levels of the image processing and understanding process.

However, very large hypercubes need many wires per PE, which makes them difficult and expensive to build. For performing image processing and understanding on large images, rather than build a hypercube with a PE per pixel (which would almost certainly necessitate a SIMD machine with bit-serial processors), it may be better to retain a simpler efficient structure for lower level image processing, and reserve a MIMD hypercube primarily for control and higher levels of image processing. The hyperpyramid attempts to do this, trying to balance flexibility and capacity against cost and design complexity constraints, utilizing architectures which have been demonstrated to be feasible with current technology.

REFERENCES

1 S. Colley, J.P. Hayes, T.N. Mudge, J. Palmer, and Q.F. Stout, "Architecture of a hypercube supercomputer", *Proc. 1986 Int'l. Conf. on Parallel Processing*, to appear.

2 W.D. Hillis, *The Connection Machine*, MIT Press, Cambridge, Mass, 1985.

3 R. Miller and Q.F. Stout, "Data movement techniques for the pyramid computer", *SIAM J. Computing*, 1986, to appear.

4 R. Miller and Q.F. Stout, "Efficient graph and picture algorithms using general data movement operations for the mesh-of-trees and hypercube networks", submitted.

5 T.N. Mudge, "Vision algorithms for hypercube machines", *Proc. 1985 Comp. Arch. for Pattern Anal. and Image Database Manag.*, pp. 225-231.

6 F.P. Preparata and J. Vuillemin, "The cube-connected cycles: a versatile network for parallel computation", *Comm. ACM* 24 (1981), pp. 300-309.

7 C.L. Seitz, "The Cosmic Cube", *Comm. ACM* (28), 1985, pp. 22-33.

8 J.S. Squire and S.M. Palais, "Physical and logical design of a highly parallel computer", Tech. note, Dept. of Elec. Eng., Univ. Michigan, Oct. 1962.

9 J.S. Squire and S.M. Palais, "Programming and design considerations for a highly parallel computer", *Proc. Spring Joint Computer Conf.*, 1963, pp. 395-400.

10 H. Sullivan and T.R. Bashkow, "A large scale, homogeneous, fully distributed parallel machine, I", *Proc. Computer Arch. Symp.*, 1977, pp. 105-117.

11 H. Sullivan, T. R. Bashkow and D. Klappholz, "A large scale, homogeneous, fully distributed parallel machine, II", *Proc. Computer Arch. Symp*, 1977, pp. 118-124.

12 L.G. Valiant, "A scheme for parallel communication", *SIAM J. Computing* 11 (1982), pp. 350-361.

This research was partially supported by the National Science Foundation, grant DCR 8507851.

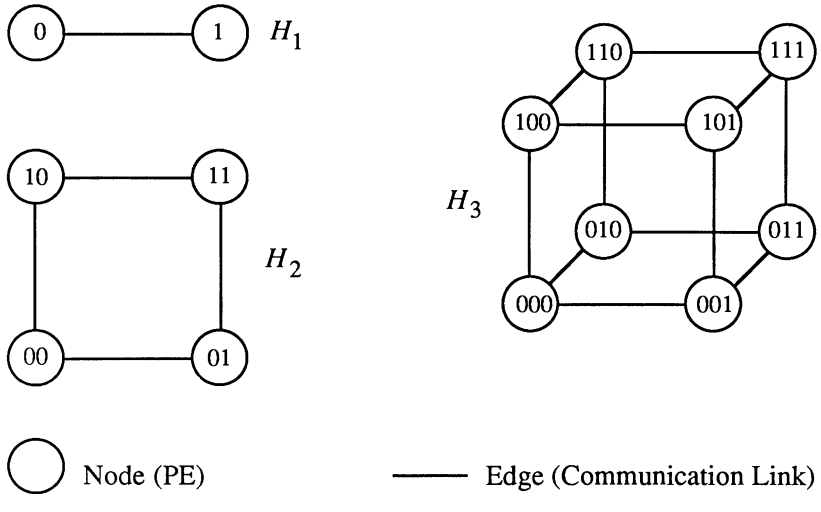


FIGURE 1. Small Hypercubes.

G_1:

0	0
1	1

G_2:

0	00
1	01
2	11
3	10

G_3:

0	000	$0G_2$
1	001	
2	011	
3	010	
4	110	$1G_2$ reversed
5	111	
6	101	
7	100	

FIGURE 2. Reflexive Gray Codes

G_2 \ G_3	000	001	011	010	110	111	101	100
00	00000	00001	00011	00010	00110	00111	00101	00100
01	01000	01001	01011	01010	01110	01111	01101	01100
11	11000	11001	11011	11010	11110	11111	11101	11100
10	10000	10001	10011	10010	10110	10111	10101	10100

FIGURE 3. Embedding a 4x8 mesh onto H_5

	000	001	011	010	110	111	101	100
000	*			*	*			*
001								
011								
010	*			*	*			*
110	*			*	*			*
111								
101								
100	*			*	*			*

FIGURE 4. Pyramid Parent PEs in Each Square

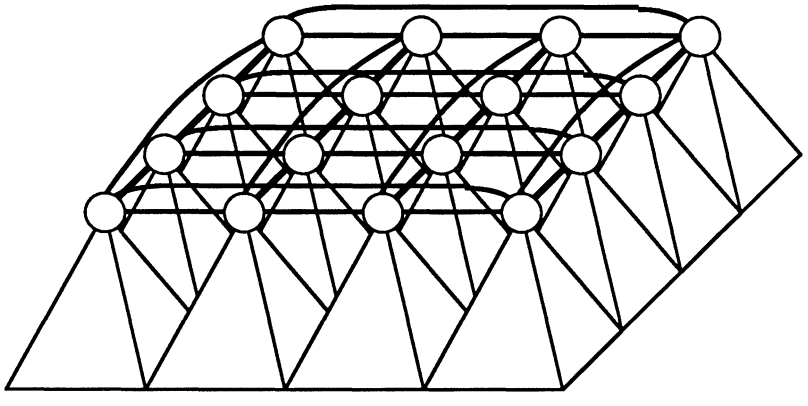


FIGURE 5. A Hyper-Pyramid

ARCHITECTURAL COMPARISONS

S. CASTAN

Lab. C.E.R.F.I.A.
I.U.T. Dept. Informatique
Univ. P. Sabatier, Toulouse, (France)

ABSTRACT

This paper explores the use of cellular processors such as pyramid machines used to process multiresolution algorithms for image processing and computer vision. In the first section we examine the image processing problems, and we determine the conditions to obtain a reasonable processing time. In the second section we examine the main parallel structures deduced from the Von-Neumann architecture. In the third section the pyramid computer is examined, we compare different mappings from the tree nodes to the processors, the type of processors and the synchronisation problems. In section four we discuss the appropriate architecture for image processing.

I. INTRODUCTION

This paper explores and sketches out ways in which parallel architectures can be used for image processing.

In image processing the volume of data and the algorithm complexity to be processed depend simultaneously on the application area and the problem to solve. Thus the number of pixels can vary from 4096 (64 * 64) in scintigraphic imagery, up to 63×10^6 pixels (3000 × 3000 in three channels (1 pixel = 20 m × 20 m) and 1 channel with 10 × 10 m per pixel) in SPOT images. Other important applications are robot vision, flexible visual inspection and flexible assembly automation. In these cases the volume of data varies from 16.384 pixels (128 * 128) to 786.432 (512 * 512 in the three channels Red Green Blue), with the real time constraint. On the other hand the various problems encountered are : image data base management, image registration, compression and transmission, description and automatic recognition of various objects in an image and understanding a given scene. In spite of the diversity of

NATO ASI Series, Vol. F 25
Pyramidal Systems for Computer Vision
Edited by V. Cantoni and S. Levialdi
© Springer-Verlag Berlin Heidelberg 1986

these application areas, computer vision can be considered as divided in two sub-problems : low-level and high-level vision.

Low-level problems concern real world image input, restoration and enhancement (filtering), segmentation, feature extraction (edge detection, size, shape, color, texture etc... parameters) and a first description of the given image.

High-level vision concerns problem-dependent knowledge using artificial intelligence technics. The symbolic description of the image can be any very high level representation of the data in terms of real world objects and concepts.

We notice there is no sharp separation between these two levels and a great interaction exists. The low-level is the pixel-level where we process pixels or set of pixels. The high-level is the symbolic level where symbolic description of the image is processed. That could imply different kind of architectures for each level and within each level.

The figure 1 shows a general vision system :

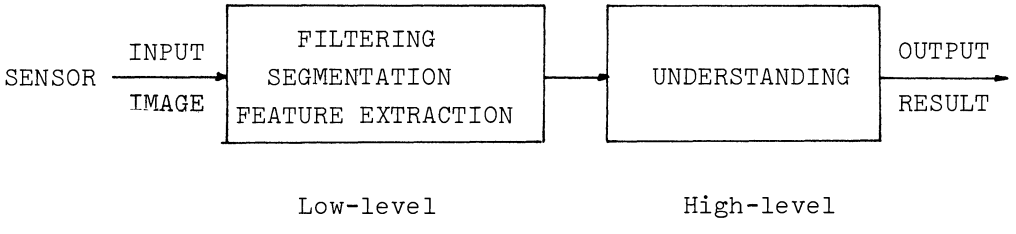


Figure 1 : GENERAL VISION SYSTEM

So image processing architectures are functions of different parameters such as :

- the problem for a given application
- the algorithms to be implemented in connection with the architecture
- the technical hardware constraints necessary with the processing time - cost ratio
- the specificity of the image data in which we have both an enormous volume of bytes to be processed and the spatial connectivity of each pixel or voxel in the two or three dimensional space.

In order to achieve reasonable processing times, we can seek to increase the power and the speed of the computing unit and/or decrease the volume of data to be processed. But in spite of recent technological progress conventional computers cannot process fast enough and parallelism has to be used. Secondly a single structure cannot manage all image processing problems with enough efficiency and flexibility, so it will be necessary to look for a multi-structure solution.

The remainder of this paper will consist of an examination of these two topics.

II. THE MAIN PARALLEL STRUCTURE

Parallel structures are often referred as non-Von-Neumann structures. This conventional general purpose serial structure is basically built with four linked modules (figure 2) :

- a memory
- a Central Process Unit (CPU)
- a controller unit
- an Input-Output devices linked to the system.

In the CPU are an Arithmetic and Logic Unit (ALU) and a set of registers. The controller unit has to synchronise all operations to be done such as fetching instructions and data stored in the memory executing the specified operations on these data, and storing the results in the linked memory. In addition Input/Output devices are linked to the system interfacing the computer and users.

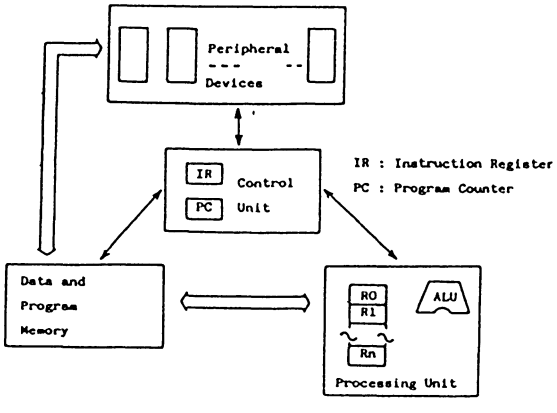


Figure 2 : The VON NEUMANN Structure

So from the Von-Neumann structure it is possible to imagine a great diversity in parallel structures, and many taxonomies have been proposed such as :

- The Flynn's taxonomy (1972) : He does not base his macroscopic classification of parallel architectures on the structure of the machines, but, rather on how the machine relates its instructions to the data to be processed. There are four broad classifications, according to whether the instructions or data stream are single, or multiple (SISD, SIMD, MISD, MIMD).

- The Shore's taxonomy (1973) : He bases his classification on how the computer is organised from its constituent parts. Six different types of machine were recognised and distinguished by a numerical designator.

- Baer's taxonomy : He suggest four classes distinguing pipeline, arrays processors and homogenous or not multi processors.

- Erlanguen's classification : established by Handler from the three operators : control unit, processing unit and logical unit.

- The Feng's classification : gives a graphical representation from the number of bits processed in parallel and the number of words processed in parallel.

The Cantony and Levialdi's classification : giving a graphical representation from correspondence with the process structure and the correspondence with the data structure.

- The Hockney and Jessope's structural classification : giving a tree description.

Unfortunately none of these classifications are satisfying. The Baer's classification takes into account the pipe-line, the Shore's classification shows off the parallelisation of structural mechanisms. The Erlangen and Feng's classification are much more measurement tools than a taxonomy. The Hocney Jessop's structural notation is more a descriptive tool than a classification. Nevertheless the Cantoni's classification is more qualitative and is interesting in image processing, and the Flynn's mnemonic such as SIMD and MIMD are well known in the computer science community, and we will use them.

In fact the main parallel structures may be regarded as a variation of the Von Neumann structure when parallelisation is performed by duplication of some part of this structure. We can obtain the following six structures (figure 3) :

a) Duplication of the Processing Unit may give a Pipe-Line structure when the results of one duplicated Processing Unit enter the next one. If the pipeline has M Processors, the progral executes up to approximately kM times as fast as one Processor Unit. The Sternberg's Cytocomputer is a pipe-line multicomputer specialised to execute image processing operations.

b) In the functional parallelism each duplicated Processing Unit is specialised for some complete treatment. The Toshiba's PPP is an example of such a structure.

c) The Control Unit may be duplicated simultaneously with the Processing Unit and give a multiprocessor structure. Up to now this type of structure has not encountered great success and the example of the Dencolor's HEP seems to confirm this failure.

d) When all the duplicated Processing Units are identical, the resulting structure is an Array Processor where many interconnection schemes may be thought of, both for Processor-Processor connections and for data memory-Processor connections. The ICL's DAP, the Duff's CLIP 4, and the MPP are well known examples of Array Processors.

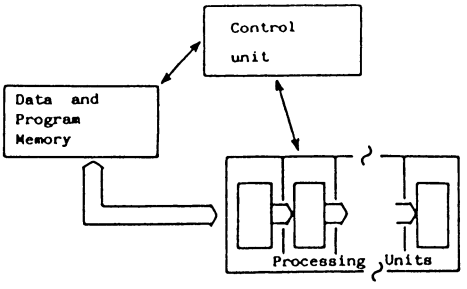
e) The use of a content Adressable Memory for the data memory in conjunction with the duplication of the Processing Unit will provide a Content Addressable Parallel Processor. STARAN is the classic example built on this concept.

f) In the data flow concept the instructions are fired as soon as their operands are available. The duplicated processing units have a control unit without a program counter.

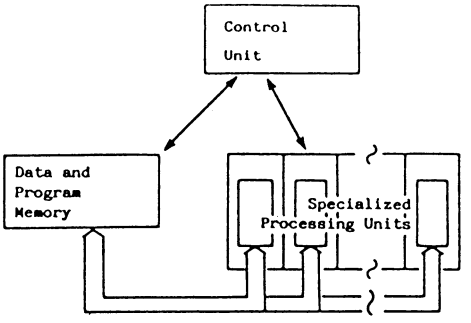
These structures have been used in different ways to increase the power of various computers dedicated to image processing.

The second way for decreasing the computing time is to reduce the volume of the data to be computer. One solution is the multi-

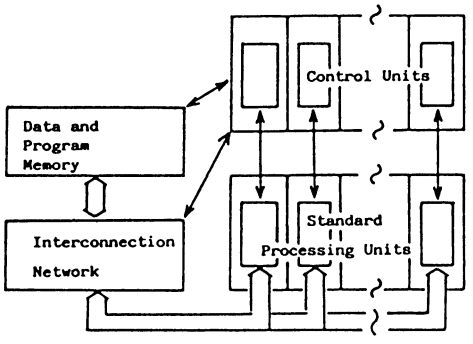
resolution approach, associated with the corresponding structure : the cellular pyramid machine.



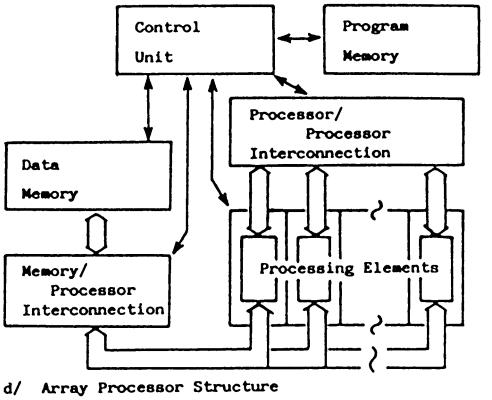
a/ Pipe-line Structure.



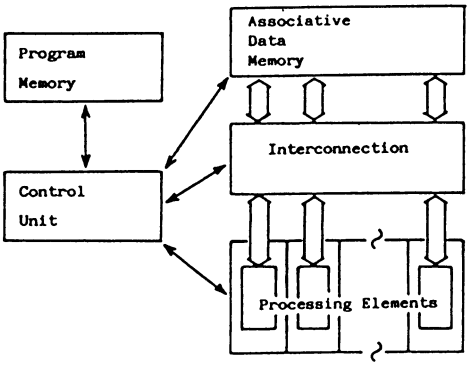
b/ Functional Parallelism.



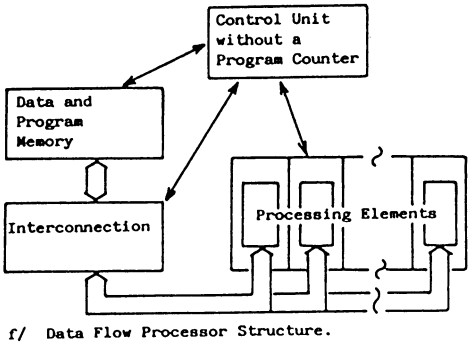
c/ Multiprocessor Structure.



d/ Array Processor Structure



e/ Content Addressable Parallel Processor Structure.



f/ Data Flow Processor Structure.

Figure 3 : Six parallel variations of the Von-Neumann structure

III. THE PYRAMID COMPUTER

Pyramid data structures or processing cones or layered recognition cones are well suited for processing at multiple levels of resolution, ensuring the use of the most appropriate resolution for a given task in an image, and giving a representation at multiple levels of resolution so that subsequent processes may make use of the most appropriate description. Pyramids have been used primarly for the purpose of efficiently computing multi-resolution features. Pyramids also provide a regular hierarchical interconnection strategy from wich various hierarchical object representations can be derived.

3.1. The pyramid image representation

The central feature of the pyramidal representation is a hierarchy imposed on the image by a recursive square decomposition of the image. This hierarchy has formed the basis of various puramids approaches, and of the quadtree representation of images. To obtain the quadtree the image is overlaid with an increasingly fine tesselation that defines a recursive embedding of quadrants. The hierarchy is described by a tree whose root node is associated with the entire image. Each tree in the node representing a window, a node has four children until it corresponds to the smallest window size allowed. Each of the four child nodes is associated with a quadrant parent window and leaf nodes correspond to pixels or to windows of the smallest size.

The choice of the mapping from the tree nodes to the processors, the type of processors and the synchronisation of each level determine the different architectures discussed in this paper. One of the principal advantages of the pyramid is that an image can first be analysed at a coarse resolution, and the results of that analysis can be used to selectively guide the application of subsequent operations to finer resolution versions of the image.

3.2. Different mappings from the tree nodes to the processors

3.2.1. The quadtree representation

The first approach assigns one processor to each node in the tree proposed by Uhr, in which a layer contains one processor

array and one memory array. Each elementary cell in a layer contains one processor and its memory store. In that case the pyramid starts with the base array of memory stores :

```
01 01 01 01 02 02 02 02
01 01 01 01 02 02 02 02
01 01 01 01 02 02 02 02
01 01 01 01 02 02 02 02
03 03 03 03 04 04 04 04     11 11 12 12
03 03 03 03 04 04 04 04     11 11 12 12
03 03 03 03 04 04 04 04     13 13 14 14    21 22
03 03 03 03 04 04 04 04     13 13 14 14    23 24      3

      Layer 0                 Layer 1     Layer 2    Layer 3
      (image)                                          (apex)
```

Figure 3 : building a pyramid from succesively smaller arrays : 4 layer pyramid with an 8 * 8 image base and a 2 * 2 convergence

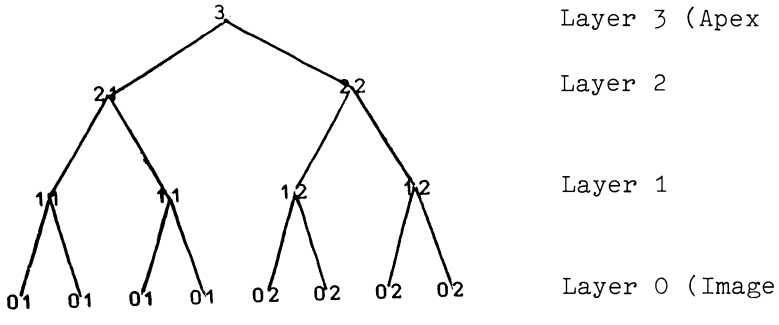


Figure 4 : Quadtree decomposition of the pyramid in Figure 3

3.2.2. Additional links

The pyramd is a "stack" of images, where the image at level k can be obtained by averaging and sampling the image at level k-1 we will assume that the bottom or 0th level of the pyramid is a 2**n*2**n image and that the k**th level is a 2**(n-k)*2**(n-k) image obtained by averaging non-overlapping windows of the image at level k-1. A pixel in the level k image is defines to be adja-

cent to :

- Its 4 or 8 neighbors at level k (like an array processor)
- The 4 pixels at level k-1
- The pixel at level k+1

For example the Tanimoto pyramid neighborhood of cell (i,j,k) is fourteen : the nine lateral neighborhoods, the father and the four sons. In that case the total number of cells in such a pyramid is approximately 33 % more than the number in its largest array. In the PAPIA in the MPP machine there are : 1 father, 4 sibting and 4 children, and in the Merigot project : 1 father, 4 sibting and 2 children.

3.2.3. An overlapped pyramid

In this case the major difference with the previous versions is in the selection of a node's neighborhood which explicitly links each node to others at its own level, the level below it and the level above it. Hanson and Riseman proposed structures with large overlapping neighborhood sizes. Burt recently has considered pyramids in which parent-children links are dynamically determined varying from iteration to iteration in an algorithm. In the Kropatsch's proposal, layer k overlap layer k-1 with 45° rotation. Dyer's proposal is the following : his overlapped pyramid can be described as the superposition of two nonoverlapped pyramids in which at each level a node in one pyramid represents a block of pixels in the original image which is shifted 50 % with respect to the corresponding node in the other pyramid. So, for a 2**N *2**N image, we define an overlapped pyramid as an (n+1)-level structure where node (i,j,k) specidies a node at coordinate (i,j) within the array at level k (level 0 is the bottom array and level n is the top or the root). Rows and Columns are indexed from the top left from 0 to (2**(n-k+1))-2, with k > 0.

Node (i,j,1) has four children (i,j,0), (i,j+1,0), (i+1,j,0), (i+1,j+1,0).

Node (i,j,k), k > 1 has five children (2i,2j,k-1), (2i,2j+2,k-1), (2i+2,2j,k-1), (2i+2, 2j+2,k-1), (2i+1,2j+1,k-1).

Node (i,j,k) has four siblings (i,j-1,k), (i-1,j,k), (i,j+1,k), (i+1,j+1,k) ; each node has four parents with k = 0, and has one parent if i or j is odd, otherwise has four parents with 0 < k < n.

Thus level 0 contains a (2**n) * (2**n) array,
level 1 contains a (2**n)-1 * (2**n)-1 array,
level k, 0 < k < n, a (2**(n-k+1))-1*(2**(n-k+1))-1 array

In that case the pyramid contains less than 7/3 as many nodes as in the original array.

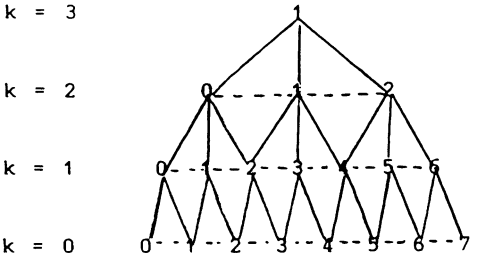


Figure 5 : An example of the Dyer's overlapped pyramid in a monodimensionnal case

The first advantage of this pyramid definition is that it provides a reduced resolution partition of the original image at each level without incurring the disadvantages of non overlapped pyramids. This structure quarantees that every pair of nodes at level = 0 which are 4(N) distance d apart, have a lowest common ancestor at level (Log d+1).

The second advantage is that this structure is less sensitive to object position and image size than in original quadtree.

3.2.4. Interleaved pyramid

A method to utilize the nonapex PEs better is to share them among several pyramids, this approach is called interleaved pyramids.

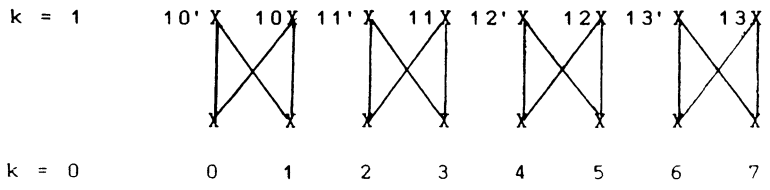


Figure 6 : Interleaved pyramid with two levels

If we make assumption (Ahuja et al.) that a PE at level k has half the workload of its parents at level k+1, it can be time-shared by two parents. Given the leaf nodes, let us insert additional nodes at level k = 1 to double their number. Now each leaf sends alternatively data to two different parents (figure 6). We notice that the same computation on tree nodes is performed both by the basic and the interleaved pyramid (only the PE assignment to the nodes changes). The algorithms for the basic pyramids are also valid for the interleaved pyramid.

3.3. Processor element hardware

Different solutions have been proposed to select the Processor Element in pyramid computers.

In the Tanimoto's proposal all cells are an identical 1 bit PE capable of performing cellular logic. Each PE is capable of transmitting and receiving information with any cell in its fourteen pyramid neighbohood. Each PE has a one-bit register, thirteen inputs with 8k memory, the chip is 4×4 PE and the bottom of the pyramid is 8×8 PE. A PE is capable of executing the following types of instructions : matching of patterns to neighborhood data storing of bits from its register into its local memory and vice-versa, and bit serial arithmetic on values in memory.

On the other hand Uhr suggests the use of different types of processors. The largest layer has 1-bit processor (in a CLIP 4 or MPP way), each of them with 1k, 4k, 16k or 64k bits of memory. At higher layers, processors will increase in size to 4, 8, 16, 32 bits.

The Davies proposal is the utilization of microprocessor rather than simple processing elements to facilitate writing programs which implement complex image analysis operations. However these more general machines have a much more complex overall organization than a cellular array. By the use of this scheme H.J. SIEGEL has been designing PASM, which contains a set of microprocessors, memories and an interconnection network which provides for communication between processors and memories.

A large number of people have been examining pyramids for their properties and potential uses, but there are very few realizations. We note the Pyramid Image Processor (PIP), a joint Boing Aerospace-University of Washington project built along the lines of Tanimoto's design. The PIP is planed to have a 128×128 array of computer at its base, it converges at the rate of 2×2 without overlap, and it is a complete pyramid and contains eight layers. A chip is constructed that will handle 64 processors (8×8 PE) and Tanimo built a machine with 8 8 PE at the bottom of the pyramid. Davis Schaefer using the MPP chip built a pyramide machine with a base of 16×16 PE. Another pyramid machine is PAPIA designed by Cantoni et al. The base of the pyramid is 32×32 PEs. A chip has been designed including five PEs, one PE being the father of the four others.

Merigot et al. have designed a 1 bit PE prototype LSI including an ALU, registers and a 64 word local memory with 7 connections (father, 4 siblings, 2 children).

Dyer predicts, that by the end of the decade we can expect a leaf chip to contain a 6-level subtree on a single chip (1365 nodes). In this case each leaf chip could store a 32×32 subarray of the original image.

3.4. The synchronization

The PE can be synchronized in different ways : most authors suggest bottom - up image analysis algorithms, in that case the pyramid can run in pipeline way each layer running in SIMD node. Others algorithms are running bottom-up and top down alternatively so the pyramid runs in multiple MIND mode with each layer

functionning in SIMD mode. On the other hand some feature extraction algorithms (for segmentation or stereo vision) do not run untill the apex.

4. WHAT KIND OF ARCHITECTURE FOR IMAGE PROCESSING

A common distinction is to be made between region-level and pixel-level treatments in the iconic domain in image processing. This leads to three types of structures :

- Specialised modules for specific treatments

More and more cheap suitable tools are avaible to complete an architecture and make it efficient for some specific treatments (Fast Fourier Transform for example).

- Multiprocessor structures for region level treatment

These structures may differ from each other according to an interconnection network and are running in a MIMD mode.

- Multiprocessor structures for pixel-level treatment

This level has been carried out in several structures. Cellular arrays are well suited to image processing.

With regard to the other domain of image processing : the symbolic one, the MIMD and SIMD are also valid. Among the cellular arrays we are inclined to line processors, and we have considered the use of such Line Processors for symbolic treatments.

Finally, this second domain of image processing also has to take advantage of High level Language Machines (such as LISP or PROLOG machine) to provide some facilities in image understanding. Figure 7 summarises these approaches.

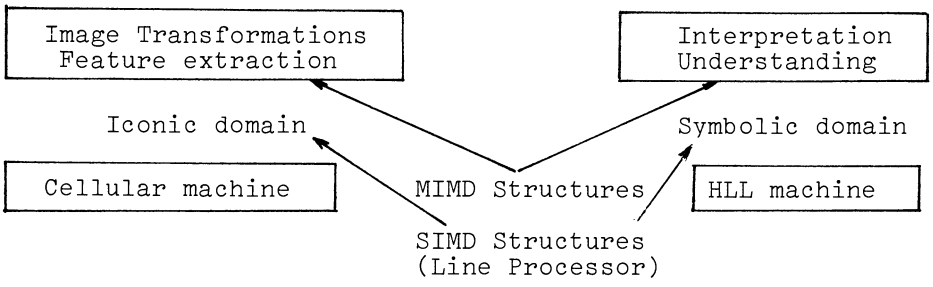


Figure 7 : Structures for the two domains in image processing

Figure 8 gives a reasonable approach to an image processing machine avaible for iconic and symbolic as well : the SYMPATI project built at the CERFIA. The MIND part has been simulated and theoretically studied ; the SIMD part has also been separately simulated and a module is presently at adjustment stage (figure 9).

With regard to the SIMD part, we adopted a one-dimensional array processor, since images are commonly projections on a 2-D plane, the two dimensionnal Array processors are of course the best choice. But two aspects have to be taken into account. The first one concerns efficiency : a good illustration can be obtained with treatments involving only some part of the whole image ; then a lot of processors have to be masked so the relative efficiency decreases in proportion. The second aspect is financial : in spite of the constant progress in VLSI technology, the cost of an array processor is limited to some huge and expensive systems.

Besides the advantages of the low cost which makes it possible to design an image processor at a reasonable price, compatible with some PC configuration, another advantage is the modularity of such structure in hardware and multiresolution approach as well.

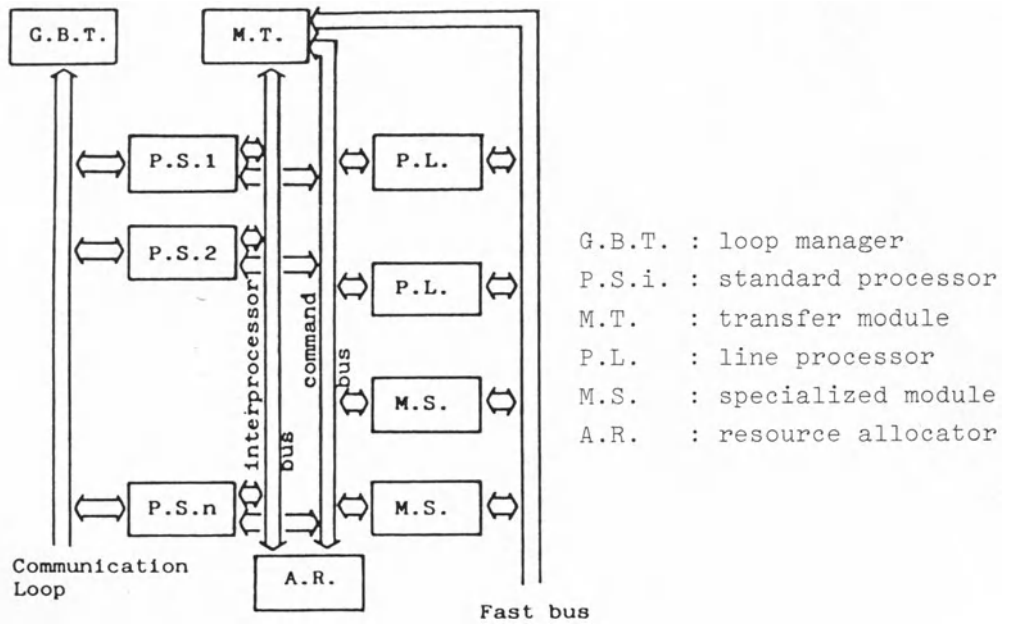


Figure 8 : The SYMPATI project structure

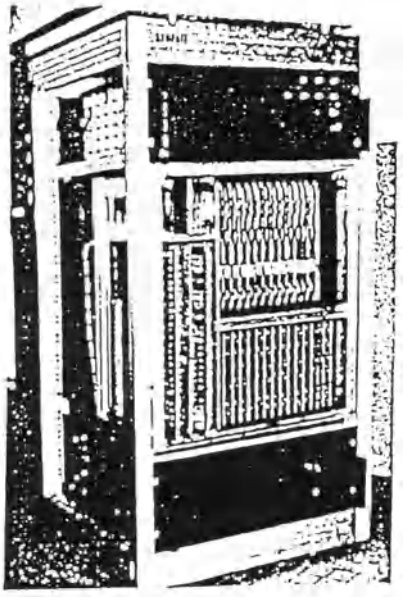


Figure 9 : A SYMPATI Line Processor

5. CONCLUDING REMARKS

Image processing is computation-time intensive, so many people have studied various specific architectures to increase the power and the speed of the computing units ; multiresolution approach is another way of decreasing the volume of the data to be processed. Pyramid machines are well suited to perform multiresolution algorithms, and offer potentially orders-of-magnitude increases in speed and power over traditional single Von-Neumann computers, and they appear to be especially appropriate for image processing. On the other hand it appears that multistructure architectures are necessary in order to satisfy the image processing domain. Such multistructure solutions should consist of three parts : some specialised modules for specific treatments, a MIMD part for the region-level treatments, and a cellular structure for the pixel-level treatments. Waiting for the low cost VLSI technology, a reasonable structure is the Line Processor which is relevant to both iconic and symbolic treatments.

6. REFERENCES

Ahuja N., Swamy S.
Multiprocessor pyramid architectures for bottom-up image analysis. In Multi resolution image processing and analysis, ed. A. Rosenfeld. Springer-Verlag, 1984.

Ahuja N., Swamy S.
Interleaved pyramid architectures for bottom-up analysis. Proc. of the 6th Int. Conf. on Pattern Recognition. Munich Germany 19-22 Oct. p. 388-390 Vol. 1, 1982.

Basille J.L., Dalle P., Castan s.
Iconic and symbolic use of a line processor in multi-level structures. In Intermediate level image processing, ed. M.J.B. Duff. Ac. Press, 1985 (to be published).

Basille J.L., Castan S.
Multilevel architectures for image processing. SPIE-2th International Symposium on Optical and Electro-Optical Science and Engineering. Nov. 25-Dec. 6, 1985, Cannes, France.

Basille J.L.
Structures paralleles et traitement d'images. Thèse de Doctorat d'Etat, Dec. 9 1985.

Cantoni V., Carrioli L., Cinque L., Ferretti M., Levialdi S., Perone A.
Image processing in hierarchical computer structures. Digital Signal Processing. ed. Cappellini V. and Constantinides A. G. Elsevier Science Publishers B.V. (North Holland), 1984.

Cantoni V., Ferretti M., Levialdi S., Maloberti F.
A pyramid project using integrated technology for parallel image processing. ed. Levialdi S. - Ac. Press N.Y. p. 121-132, 1985.

Castan S.
Parallel processing. In Computer architectures for spatially distributed data. ed. Freeman H.T., Pieroni G.G., Nato Series Springer-Verlag, p. 349-374, 1985.

Davies L.S.
Computer architectures for image processing. Proc. of the worshop on picture data description and management. Asilomar, CA, USA, 27-28 Aug. 1980, p. 249-254.

Duff M.J.B.
CLIP 4. In Special computer architectures for pattern processing. ed. Fu K.S. and Ichikawa Ac. Press, 1982.

Devos F., Merigot A.
Integration d'un processeur cellulaire pour une architecture pyramidale de traitement d'image. Revue de Phys. Appl. 20 (1985) 23-27.

Dyer C.R.
Pyramid algorithms and machines. In Multicomputer and image processing. ed. Preston K, Jr and Uhr L. p. 409-412, 1982.

Miller R., Stout Q.F.
The pyramid computer for image processing. 7th. International Conf. on Pattern Recognition. Montreal, QUE. Canada 30 July-2 Aug. 1984, p. 240-242.

Merigot A., Zavidovique B., Garda P., Devos F.
Pyramidal algorithms for image processing. 7th. International Conf. on Pattern Recognition. Montreal, QUE. Canada, 30 July-2 Aug. 1984

Schaefer D.H.
A Pyramid of MPP Processing elements. Experience an plane. Proceeding of the 18 Annual Hawaï international Conference on system Science, Jan. 1985.

Tanimoto S.L.
A pyramidal approach to parallel processing. 10th. Annual Int. Conf. on Computer Architecture conference proceedings. Stockholm, Sweden 13-16 June 1983, p. 372-378.

Tanimoto S.L.
Graphical operations in a hierarchical parallel computer. Proc. SPIE Int. Soc. Opt. Eng (USA) Intelligent robots and computer vision. Cambridge, MA. USA 5-8 Nov. 1984, vol. 521, p. 195-201.

Uhr L., Thompson M., Lackey J.
A 2-layered SIMD/MIMD parallel pyramidal "array/net". 1981 IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and image Database Management. Hot Springs, VA, USA 11-13 Nov. 1981, p. 209-216.

Uhr L.
Converging pyramid arrays. 1981 IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and image Database Management. Hot Springs, VA, USA 11-13 Nov. 1981, p. 31-34.

Uhr L.
Comparing serial computer, arrays and networks, using measures of "actives ressources". 1981 IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and image Database Management. Hot Springs, VA, USA 11-13 Nov. 1981 p. 188-194.

Uhr L.
Augmenting pyramids and arrays by compounding them with networks. 1983 IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and image Database Management Proceedings. Pasadena, CA, USA 12-14 Oct. 1983 p. 162-168.

Uhr L.
Feasible multi-computer architectures, given 3-dimensional stackerd wafers. 1983 IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and image Database Management Proceedings. Pasadena, CA, USA 12-14 Oct. 1983 p. 80-86.

Uhr L.
Algorithms-Structured computer arrays and networks. Ac. Press 1984.

Uhr L.
Multicomputer parallel arrays, pipelines, and pyramids for pattern perception. VLSI and modern signal processing p. 406-421.

A PYRAMIDAL SYSTEM FOR IMAGE PROCESSING

A.Merigot*, P. Clermont**, J. Mehat*,
F. Devos*, B. Zavidovique***.

* IEF - Universite Paris Sud - 91405 Orsay - France
** ETCA - 16bis Av. Prieur de la cote d'or
94114 Arcueil- France
*** ERL - 292 Cory Hall - U.C. Berkeley - CA 94720 - USA

1. Introduction

Because of the large volume of data handled during image analysis, a high degree of parallel processing is required in special purpose image processing computers. Cellular arrays, as, for example, CLIP4 [Duff 76], MPP [Batcher 80], DAP [Flanders 77] can realize very efficiently pointwise or local operations on images, thanks to their mesh organization, but these structures lack of efficient and flexible global data extraction mechanism to perform higher level tasks such as feature extraction, object detection, and so on....

To go beyond these limitations, and perform efficiently multi resolution image processing [Uhr 72, Rosenfeld 84], pyramid machines are currently under development in several places [Tanimoto 83a, Cantoni 85ab, Scheafer 85]. Their architecture is based on a hierarchical network of processing elements, connected as layered stacked arrays of decreasing size. Each cell is connected to four or eight neighbors in its layer, and to a parent cell in the upper layer and four children cells in the lower one.

We present in this paper an original pyramidal structure which is under development as a joint project of the Institut d'Electronique Fondamentale CNRS laboratory in Paris Sud University, Orsay, and ETCA defense research lab in Arcueil, France. It is composed of a binary tree of processing elements whose sons are alternatively of the same X and Y coordinate, to ensure enough isotropy in the pyramid organization (figure 1). Thus every PE can communicate with its father, its two sons, and four neighbors inside its layer.

In section 1, we describe the internal organization of the processing element (PE). Section 2 presents the key motivations which leaded to the choice of a binary structure, and the way it can be used to perform several basic operations. Section 3 details the layered MIMD working mode of the structure, and the way it will be implemented by means of a distributed control.

NATO ASI Series, Vol. F 25
Pyramidal Systems for Computer Vision
Edited by V. Cantoni and S. Levialdi
© Springer-Verlag Berlin Heidelberg 1986

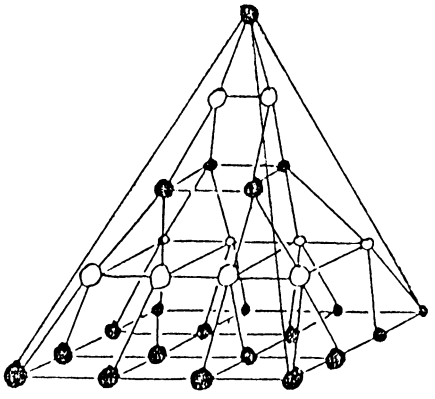


FIGURE 1. ARCHITECTURE OF SPHINX.

2. The processing element

With the actual integrated technology, a 128 by 128 pyramid machine seems to be a reasonnable goal within a few years. To succeed in this realization, the individual processor has to be as small as possible. This can be achieved by means of bit serial processing and bit serial data transfer to reduce the silicon area and IO pin count. These choices had yet been made in cellular meshes and for pyramid machines, because of the large number of inter processing elements communication path required, the number of IO pins is several order larger, and the use of a bit serial PE is the only way. This IO pin count limit leaded us to choose on chip memory. Thus, to increase the number of processors available in a single chip, we decided to limit the memory size to 128 bits, and to associate to every pyramid chip a standard memory chip with long access time but large storage capacity.

The architecture of the PE is in given in figure 2. It consist in an ALU, some special purpose registers and communication pathes, and a 128 one-bit-word dual port memory.

2.1. The arithmetic and logic unit

The ALU can perform any of the 16 logical operations on its two input A and B, and several arithmetical operations between the A and B input, and the carry register Ca :

operation	ALU output	Carry
ADD	A ⊕ B ⊕ Ca	A*B+A*Ca+B*Ca
SUB	A ⊕ /B ⊕ Ca	A*/B+A*Ca+/B*Ca
NADD	/(A ⊕ B ⊕ Ca)	A*B+A*Ca+B*Ca
NSUB	/(A ⊕ /B ⊕ Ca)	A*/B+A*Ca+/B*Ca

With these operations we can perform most of the arithmetical operations. The NSUB operation is used to compare to n bits integers in n steps. With a constant operand explicitely accessed thanks to pointers, (see below), the ADD and SUB operations can perform as well one argument operations such as increment, decrement,2's complement...

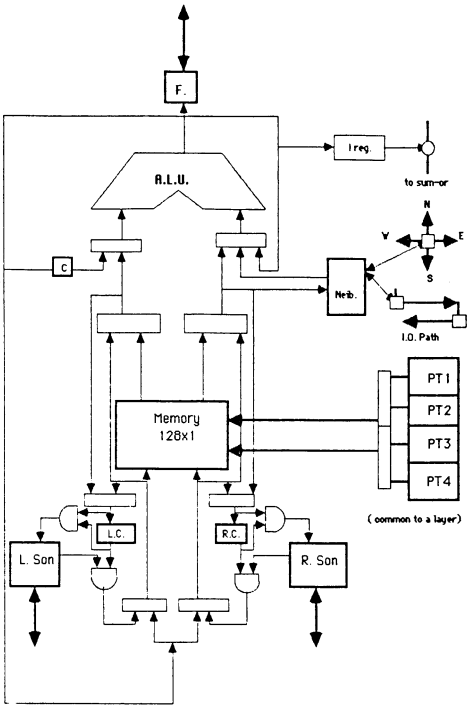


FIGURE 2 - ORGANIZATION of the PROCESSING ELEMENT

2.2. The registers

We describe now the one bit wide special purpose registers.

The condition register C is used to inhibit (mask) a processing element operations, according to its content, in order to implement if...then...else... control structures. This masking capability can be either used or ignored, according to the instruction. If C is set, the PE is inhibited, ie no result is written back in the internal memory. The C register is realized with a master slave flip flop, allowing its own conditional loading, performing thus an implicit AND operation with the output of the ALU.

The carry register Ca is implicitly used as a source and a destination by arithmetical operations. It can be loaded and stored, but cannot be used as an argument for a logical operation.

The interrupt register I is used to centralize a boolean information on a single layer, in a sum-or-tree like mechanism. It is a write only register.

2.3. The vertical communication channels

The vertical communication channel is a bidirectional connection between PEs in consecutive layers. This channel can only support a monodirectional transmission at every moment to avoid the duplication of the physical communication path involved by full duplex capabilities. To improve layered MIMD algorithms programming, and to allow asynchro-

nous data transfer between the different layers [Merigot 86a], the vertical communications are buffered through mail-boxes registers. Thus sending data to another layer means writing in a local buffer register, and to receive this data, a layer read this register in the emitting layer. The control of the data validity in these registers and of the transmission direction of the shared inter layer communication path is made externally at the distributed control level.

Additionally, these communications can be masked by the Sons Control Registers SCRs. The data transmitted through the channel (in both directions) are ANDed with the content of these registers.

2.4. The neighbor channel

In the first version of SPHINX, the neighbors communications were realized through a direct access to the neighbor's memory. This system was very flexible, but finally, this channel transfert rate determined the PE working rate, slowing it down. In the actual version, the channel is accessed through a four direction shift register which can works concurrently with the other parts of the system.

2.5. The internal memory

SPHINX has 128 bits of internal memory which can be accessed by two independent busses. This allows a dual read-modify-write operation with two arguments from the memory. The results can be writen back in one or two location of the memory. It is also possible to write two different data in two places of the memory - the two sons for instance, or the output of the ALU and the neighbor - or to write concurrently in the memory and in a communication channel. These capabilities are useful to make on-the-fly copies of a remote argument, which has to be used several time, as it happens in convolution for instance.

This memory is accessed by means of a set of pointers. These pointers are common to all the PEs in a layer. In the current architecture, there are four pointers and they can be independently incremented, decremented or left unchanged. The actual use of these pointers will be described now.

2.6. The pointers

Thanks to its dual port memory, the PE is a three address machine. The pointers are used in an indirect addressing way to avoid the emission of three independent addresses, expensive in terms of IO pins. Thanks to their autoincrement capabilities, they are especially suited for multibits operations. As most of these operations consist in an iterative sweep of the successive bits of the memory, they are performed efficiently with increment/decrement primitives. Here are two trivial examples:

addition of two n bits integers

```
/* reset the Ca register if necessary */
load pt1 with lsb of argument1
```

```
load pt2 with lsb of argument2
load pt3 with lsb of destination
loop n times
    *dest++ = *arg1++ ADD *arg2++ ;
endloop
/* store or test Ca to prevent overflow */
```

3 way comparison of two n bits integers

```
/* the C register is supposed to be set */
load pt1 with MSB of argument1;
load pt2 with MSB of argument2;
Ca = 1;
loop n times
    C = arg1-- NSUB arg2-- conditionally
endloop
/*
        C       Ca
        1       x       equality
        0       1       arg1 > arg2
        0       0       arg1 < arg2
*/
```

The use of pointers to access the memory presents several advantages. First, it is a simple and cheap way, in terms of control load and IO pins, to implement very efficiently multibit operations on a bit serial processor. Table 1 presents a comparison of the execution time of basic operations on 3 types of machines : a dual port memory accessed through pointers 3 address machine, the same machine with direct addressing, and a single port memory two address and one address machine. In the later cases, we supposed that the machine had flexible accumulation capabilities with a variable length shift register (such as in MPP or PAPIA).

Of course, the fastest method is the second one, but the overhead due to the pointers loading is small (less than $\simeq N$ extra time) compared to the extra cost of direct addressing; this is not true for boolean operations (N=1) : the execution time seems to be multiplied by 4, but, in this case the pointers are used in a static way, and in general it will be possible to perform several logical operations without reloading them.

The performances of the first method and the third and fourth ones are comparables, thanks to the variable length shift register. Let us now stress some advantages of the dual port memory architecture. First, the flexibility of the pointers manipulation - compared to a shift register - allows the direct adaptation of bit serial algorithms to any size variables. Thus, for instance, it is possible to reduce the execution time of the basic routines if we adapt the length of the variables to their dynamic range (for instance, the incidence of the truncation to N bits of the multiplication is much more important with the first method, and so would be the use of non-standart length variables - for instance 10 bits). Additionally, this

TABLE 1
COMPARISON OF PERFORMANCES OF BASIC OPERATIONS ON DIFFERENT TYPES OF MACHINES.

	3 addr. w. pointers	3 addr. direct	2 addr. direct	1 addr. direct
Bool. Op.	4	1	2	3
Mem. Move	N+2	N	N	2N
Increment	N+1	N	N	N
Addition	N+3	N	2N	3N
Mult.(2N)	N^2+3N+1	N^2+N	N^2+2N	N^2+3N
Mult. (N)	$\frac{N^2}{2}+\frac{7}{2}N+2$	$\frac{N^2}{2}+\frac{3}{2}N$	N^2+N	N^2+2N
minimum IO pins control	6	14	7	7
cost	low	high	high	medium

adaptation can be performed quite easily - even at high level language compile time - and should also be used to save space in the memory. Another interest of this method is to reduce the control load by suppressing the address generation operation at every instruction.

The main drawback of the dual port technique is that it imposes the use of in-chip-memory instead of standard memory circuit. But, in so far as in-chip memory is a necessity for IO pins limitation reasons, the different implementations lead to comparable silicon area. As a matter of fact, a dual port memory is about 25% larger than a classical one, due to the double access bus in the memory. Thus the overhead is comparable to the one caused by a variable length shift register for which the silicon occupation of a bit is a bit smaller than twice the area necessary for a bit in a classical memory storage.

2.7. The IO channel

To assure a speed IO transfert rate to and from the pyramid, we shall use a unidirectional shift register, as in most of cellular arrays. This communication channel will also be used to attach to every pyramid chip a standart memory chip which will be used as a secondary storage. This memory will behave as a virtual memory, with swap-in swap-out transfer mechanism. Because of the serial shift data transfer , the access time will be determined by the pyramid chip smallest size (4 in the implementation we are planning). Thus, we can have at the same time a high speed internal memory, to assure a fast operating rate, and a large and cheap external memory to store a large amount of data (typically 2 to 4 kbits per PE). The internal memory size was determined so that the swapping overhead is negligible compared to the internal data processing time.

3. The binary tree organization

In SPHINX, the processing elements are interconnected according to a binary tree structure. We claim that this structure, despite some disadvantages compared with the quad pyramid organization [Merigot 85], is more efficient for global features extraction than the quad pyramid whilst giving comparable performances for multi resolution quad tree processing.

3.1. Global features extraction

One potential advantage of a pyramid computer over other types of parallel computers is its ability to centralize global information from the bottom of the pyramid, in a time which grows logarithmically with the size of the image. Examples of useful global features which can be centralized that way are the mean of an image, its median, histogram, connected component count, and so on... Before describing different techiques, we shall distinguish three types of extraction according to the subset of the elements which is concerned. [Merigot 84]

Fisrt, if all the elements are concerned, we are in the case of a non-selective global data extraction. The data stream will flow upward through every layer, and the operation is defined by the data reduction performed on every individual layer. In general the global operation is expressed in terms of a dichotomic decomposition of the global operation on two elements.

We can also consider operations on a subset of the PEs which defined by the occurrence of the elements in the tree - for instance a specific row of pixels, a strip, a square area, and so on... In this case of implicit selection, the global function will be divided in two different types of operations which will take place simultanously in the tree : a selection operation which simply consist in the transmission to the father of either the left or right son value according to the selected subset, and reduction operation which sends to the father a function of the value of the sons. Figure 3 presents some types of subsets, and the way the feature extraction can be performed. The different operations are noted L (R) for left (right) sons selection, and X for the reduction operation.

At last, we can perform feature extraction on a subset of the image defined explicitly by the value of some

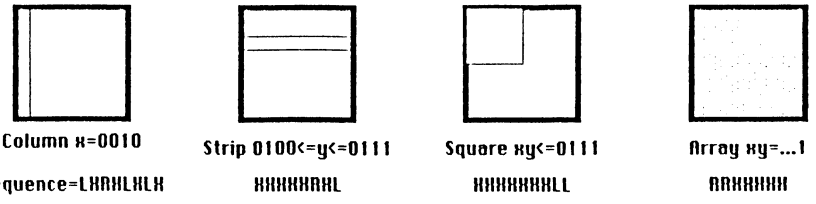


FIGURE 3 Example of implicitly defined regions

attribute associated to the pixels (for instance, a labelled region, a curve, etc...). We suppose that this subset is defined by a boolean variable. To perform efficiently this kind of operation, we can use the Sons Control Registers. According to the content of this register, the value effectively read by the PE is either the value which has been transmitted by his son, or a zero. We can use it to perform explicit selection. Indeed, if the boolean operator performed on every individual layer has zero as neutral element, ie f(X,0)=X, it will behave as a select operation if one of the sons is discarded. It effectively happens if this operation is either OR, or XOR (ADD), which are used for the more useful features (see below). Thus, to execute a feature extraction on an explicitly defined subset of the elements, we first have to build a subtree in the pyramid which leads from the apex to every selected pixel (that is an OR pyramid to retain the terminology used in [Tanimoto 83b]), and store it in the sons control registers; then perform the extraction operation; if the differents boolean operations involved in the feature extraction fits the former constraints, then the global feature will automatically be extracted from the specified subset.

One important thing to notice is that the selection operation, whether it is implicit or explicit, has no incidence on the execution time of the operation.

3.1.1. Integration

A very classical global measure examples is the summation of an attribute of the different pixel of an image (that is integrating this attribute over the surface of the image). According to the attribute which is used we can extract a large number of useful features: Integrating a boolean value - bit counting - leads to very useful features like area and circumference calculation, histograms, median computation, connected component counting, etc...; integer and real number calculations allows the computation of means, variance, moments, and so on...

This operation can be executed very efficiently on a binary tree of adders. The method is quite straighforward. To perform the addition of a value located in its two sons a cell has just to get serially the data, LSB first, and sum these bits as they arrive. As for a local addition, the first sum bit generated will be the LSB if the result, and the first generated carry bit happens to be of the same logical weight than the next bit to come, thus the process can be iterated as necessary. As a final step the carry will be the MSB of the result. To perform a global integration, one just has to iterate this process on the successive layers, send every bit of the result on every layer as they are generated to the father instead of storing it locally. Thus the overall operation is executed in a pipeline way on the successive layers. As we can see, if the initial value is coded on n bits integer, the operation on the next layer will take n+1 steps, and so on. The temporal aspect of this procedure is shown in figure 4. where we present layer occupation vs time. If h is the number of layers in the

pyramid, the total time of the operation is h for the rising slope of the operation (pipeline set-up time) and n+h to output the result. Thus the execution time is n+2h for a unique operation and n+h for multiple integrate operations performed in a pipeline way.

We can notice some useful capabilities resulting from the binary structure. Before performing the integration, if we set the carry registers of some PEs to one, in any place of the pyramid, their number will automatically be added to the result of the integration operation. For instance if every connected component in the image can be tied to a unique PE anywhere in the pyramid, to count the number of connected components in the pyramid, just set the carry in these root PEs, then perform an integrate operation on a null value...

Another application is to perform an automatic threshold on an integrated result. Thresholding a n bit integer can be performed in roughly n steps. But it is equivalent to test if an integer i is larger than a threshold k, or to test if $i+2^n-k$ is larger than 2^n, and this can be performed in a one step access to the carry. If there is 2^n PEs in the lower layer, they are 2^{n-1}, 2^{n-2}, ..., 2, 1 on the successive upper layers. Thus, setting a one in all the PEs in one or more layers is a way to code any number between 0 and $\sum_{0}^{n-1} 2^i = 2^n-1$ with distributed boolean value, on the digital basis formed by the successive layers. Thus to integrate an attribute and treshold it to k :

Compute 2^n-k.

Dispatch this value to the lower layers, so that the MSB is affected to the layer just after the lowest one, and the LSB to the pyramid apex.

Load this binary value in the carry register.

Integrate the pixel attribute to threshold.

Limitations of this procedure are the following. First, as the initialization procedure is long enough, this method is especially interesting if the same threshold has

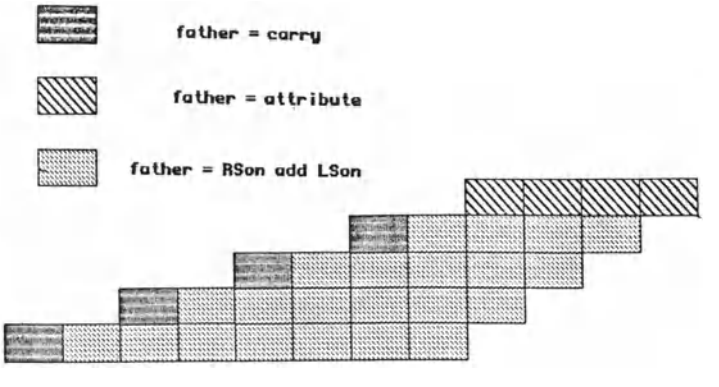


FIGURE 4 - Temporal aspect of an integration execution

to be applied several times. Second, the threshold need to be smaller than 2^n or 2^{n+1} if we use the lowest layer too. This may not be sufficient if the threshold is related to an integer value on the lowest layer (for instance a local error which we want to integrate and threshold in an optimization procedure).

This method works perfectly for median computation. A classical way to perform that operation is to threshold at the lowest layer and to adapt dynamically the successive bits of this threshold, so that half of the elements are larger than this treshold [Forster 76, Tanimoto 83c].

```
/* b bits n elements */
thresh = 0
for i = b - 1 to 0
   if count ( elements > thresh + 2^i ) > n/2
       then thresh = tresh + 2^i
   endif
endfor
median = inf ( element > thresh ? element : + oo)
```

If n is a power of two the comparison is simply the test of the MSB of the count. But in general, for instance if we want to compute the median of a segmented region, it requires a n bit comparison that can be suppressed by a proper initialization as described above.

3.1.2. Maximum computation

An other example of a global feature is the maximum of an attribute over the entire image, and others derived features (minimum, next after...); these techniques allows a fast retrievial of interesting points in an image. This maximum extraction can be computed very effectively with SPHINX. To compare these values we must start from the MSB. Initially, we mark the values as equal. Then for the successives bits, we compare them and if they differ the son with a bit set to zero is discarded. Afterwards, we send the only remaining son data. To discard a value, we reset the Son Control Register, thus the value will be replaced by zero. Thus to determine the largest of the 2 sons, we just have to OR their value, and if they differ to write back their value in the Son Control Register.

```
Father = LSon Or RSon
Condition = LSon Or RSon
SonControlRegisters = Sons conditionally
```

This operation takes 3 steps to produce every output bit and its set up time is equal to the height of the pyramid.

3.2. Emulating a quad pyramid with a bin pyramid

A pyramid machine is a natural architecture for multiresolution image processing. The SPHINX organization allows a direct implementation of bin pyramid data structure. This kind of structure has been proposed by some authors as a way to get a smaller surface reduction between two consecutive layers [Pavlidis 82]. In a binary pyramid

this reductin is 1:2 vs 1:4 for quad pyramids. The main problem which is attached to this type of processing lies in the anisotropy of the reduction function and on the metric distorsion which it generates on every two layers.

Quad pyramid processing can be very easily implemented on a bin pyramid architecture with a very small overhead. The reduction function in a quad pyramid can generally be expressed in terms of a binary decomposition of two reduction function in two consecutive layers. For instance to perform an AND (OR) pyramid, AND (OR) the two sons on the intermediate level, then AND (OR) this result on the next level. Thus this operation can be performed in 2 steps. A XOR pyramid can be computed in this way in 4 steps by XORing the previous AND and OR pyramids. A count of n pyramid (ie a pyramid where every PE is set to one if n (or more) of its sons are set) can be generated with the count-and-treshod method described above. For a count-of-2 pyramid, set the carry register in the intermediate level. For a count-of-3, set it in the upper layer; then count the number of sons. The count-of-n pyramid will be in the carry register.

Thus, for binary quad pyramid generation, the overhead on the SPHINX system, compared to the overhead on quad pyramid machine with gating in smaller than 2. Anyway a boolean pyramid generation is not the most time consuming task in multiresolution image processing. For multibit pyramids (integer or real), the time is generally speaking equivalent, as these machines have no 4 sons arithmetic capabilities. Transmitting a n bit value from the son to the father is n steps in these machine and n+1 in SPHINX, thanks to the pipeline effect, and the arithmetic operations are generally speaking more easily performed on a binary tree of adders (see above).

3.3. Other operations suitable for implementation on a binary tree

Except the operation described above and the different derived global feature extraction, we can use the tree computing structure in a pyramid machine for other operations in image processing, than strictly pixel based methods. The parallel execution of these applications have not yet been very extensively studied yet, but some have been proposed, for instance parsing [Foglein 86], implementing a priority queue [Merigot 86b], tree traversal and so on. For these applications, where the geometric location has no meaning, binary tree organization seems to lead to better efficiency than quaternary trees. We think that with the availability of pyramidal parallel machines, a larger number of algorithms suitable for this kind of structure will be studied.

4. The layered MIMD control mode

In SPHINX, the different layers are independently controlled by separated controllers, in order to achieve layered MIMD (also named Multi SIMD) working mode. This mode 's main interest is to speed up the execution of operations involving several layers concurrently. The maximum speed up which we can get is of course equal to the number

of layers, ie 15 for a 128 by 128 pyramid. there are several cases where it is possible to take advantage of this extra level of parallelism.

First, during the execution of a multi layer routine, as for instance a global data extraction. Most of these routines perform the following operations:

```
get the data from the sons
perform some amount of local computation
            and data arrangement
send the transformed data to the father
```

It is possible in many cases to send as soon as possible the transformed data (or part of it) tothe father, so that we could overlap the end of a process on a layer and the beginning of the next one. For instance, in the case of the computation of the maximum which is described above, for every bit we perform the following operations :

```
get the bit from the sons
determine the next bit of the max of the sons
                  and send it to the father
mark locally the origin of the largest son
```

Thus, the total time of the operation is determined by the duration of the partial operation instead of the total operation on a layer, thanks to the pipeline effect.

Another way to use the multi SIMD capabilities profitably concerns the parallel execution of independent routines on different layers. In that case, during the execution of a procedure, we will take advantage of idle state of a layer to run on it a routine which is formally independent of the previous one. This kind of parallelism can largely improve execution time of the machine. Let us take as an example the computation of the histogram of an image. The algorithm is quite straightforward:

```
for i=0 to max_gray_level
   select the pixel where gray_level = i
   count the selected pixels
endfor
```

In the case of this algorithm, we have first a local computation on the lowest layer followed by a count operation on the upper ones. Without using layered MIMD capabilities, the execution time for every gray level is equal to the sum of the select time, the setup time of the integration pipeline, and the result output time. With MIMD possibilities, the successive routines can collapse to reduce the execution time to the result output time. This is illustrated in figure 5 where we present layer occupation versus time in this operation.

Thus layered MIMD control can improve execution time, either inside a single routine, or for parallel execution of independant routines. The improvement factor ranges from $1+\varepsilon$ to the number of layers. The former case happens if the instruction matrix is highly repetitive (as it is for integration), and if we take into account the pipeline set-

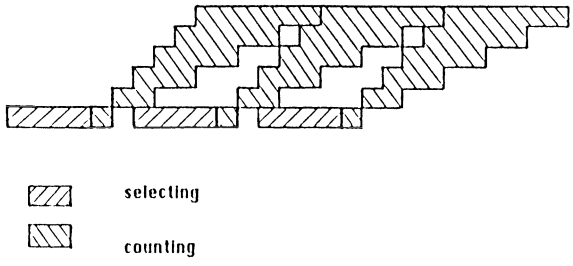


FIGURE 5
Instruction collapsing in the histogram computation.

up time. The maximum improvement takes place for irregular matrix, either in terms of instructions, or in address generation. The average amelioration should be of about 5 [Merigot 86a].

At last, the MIMD capabilities can be used directly at the algorithm level. For instance, a parallel multi resolution algorithm can be designed by providing independant local optimization operations on the different layers, with asynchronous data exchange based on a layer convergence criterion.

```
build an initial multiresolution data structure
while the global stop condition is not satisfied
      /* for instance stability in the bottom level*/
   do in parallel on all the layers
   get the context from the father
   repeat
      perform one step of local transformation
            new_state=f(current_state,context)
      if stable(new_state,current_state)
            /* sufficient stability on this layer */
         then send a new context to the sons
      endif
      current_state = new_state
   until a new context is available from the father
endwhile
```

This kind of algorithm should bring a very efficient synergy between local and global computation to prevent worst cases, and improve global result product time, though there should be many difficulties concerning the different stop conditions (that should vary according to the layer and maybe the algorithm evolution time), and generally speaking the algorithm control. The asynchronous data transfer , as well as the variing stability criterions, render this kind of algorithms not suitable for SIMD implementation.

4.1. The control strategy

The real time control of this kind of structure is a challenge as it would easily slow down the pyramid machine.

Indeed, we have to achieve a complex synchronisation between layers, within a time comparable to one pyramid instruction execution!

To realize it, we based the control of the structure on a network of controllers, one for each layer. Each controller can communicate with its two vertical neighbors (father and son) and with the intermediate host (IH). The function of the control system are the following:

* Reduce the size of the instruction code between the IH and the pyramid. For code compression, we shall use very classical methods: loops (especially useful in a bit serial system), and subroutines (some of them may even be resident in the control system). Additionally, as most of the algorithms will be almost identical on the different layeers (or at last on a significative number of layers) we shall send them only once to the controller network, in order to reduce the bandwidth.
* Insure the PE memory management, ie memory allocation, secondary storage management, swap in and out of the data and so on... The memory allocation has to be as transparent as possible to the application programmer.
* Synchronize the routine schedule on the entire pyramid. Some methods are availables [Jadbane 86]. As this operation has a coarse granularity, the associated overhead won't be too much constraining.
* Synchronize the data exchange between layers. To realize it efficiently, the buffering of the data through mailbox registers is a very crucial point for performances as well as for control implementation and software development ease [Merigot 86b]. Thus, the high level control will generate instructions in a fifo, independently of the other layers behavior, and a low level synchronization part of the control system will check for the validity of the exchanged data, and to the absence of data collision on the unique communication path, before effectively sending the instruction. This will be based upon a very simple producer-consumer-like protocol which can be implemented very efficiently with cellular automata [Clermont 86] to fit within the execution time of an instruction.

5. Conclusions

Pyramid machines present advantages over arrays for their ability to process efficiently multiresolution images, and to extract global features. The binary pyramid organization used in SPHINX, whilst having good performances to emulate quad pyramid data structures, is more flexible for global data manipulation. The MIMD control mode improve the execution time by taking into account several layers of parallelism: SIMD inside a layer, and MIMD for cooperative interlayer computation, or for parallel independant routines execution. Thus, the structure efficiency can be close to one, by preventing processor idleness.

Acknowledgements

The authors thank Patrick Garda for many helpful discussions and suggestions. This work was granted by French Ministry of Defensés DRET, and and by CNRS Silicon IC Groupement.

REFERENCES

[Batcher 80] Batcher K.E., "Design of a Massively Parallel Processor", IEEE Trans. on Computers, Vol. C-29, No. 9, pp 836-840.

[Cantoni 85a] Cantoni V., Ferretti M., Levialdi S., and Maloberti F., "A pyramid project using integrated technology", in Levialdi S. (ed.) "Integrated technology for parallel image processing", Academic Press, 1985.

[Cantoni 85b] Cantoni V., et al., "The PAPIA image analysis system", 2nd Int. Techn. Symp. on Optical and Electro-Optical Applied Science and Engeneering, Cannes 1985.

[Clermont 86] Clermont P., Merigot A., "Synchronisation de l'echange de donnees dans un multiprocesseur a l'aide d'un automate cellulaire", Internal report.

[Devos 85] Devos F., Merigot A., Zavidovique B., "Integration d'un processeur cellulaire pour une architecture pyramidale de traitement d'images", Rev. Phys. Apll., vol. 20 (1985), pp 23-27.

[Duff 76] Duff M.L.B., "CLIP4, a large scale integrated circuit array processor", Proc. 3rd ICPR, 1976, pp 728-733.

[Flanders 77] Flanders P.M., Hunt D.J., Reddaway S.F., Parckinson D., "Efficient high speed computing with the Distributed Array Processor", in "High speed computer and algorithm organization", Kuck D.J., Lawrie H.L., Sameh A.H. ed., Academic Press 1977.

[Foglein 86] Foglein J. "Systolic tree automata and their use in image processing" Communication to the Pyramidal Systems for Image Processing Workshop held in Maratea, Italy, May 1986.

[Forster 76] Forster C.C. "Content addressable parallel processors", Van Nostrand Reinhold 1976.

[Jadbane 86] Jadbane A., "Etude et conception d'un reseau lineaire de controle du processeur pyramidal SPHINX", These 3e cycle, Universite Paris Sud, Avril 86.

[Merigot 84] Merigot A., Zavidovique B., Garda P, Devos F., "Pyramidal algorithms for image processing", Proc 7th ICPR, Montreal 84, pp

[Merigot 85] Merigot A., Zavidovique B., Devos F., "SPHINX, a pyramidal approach to parallel image processing", Proc CAPAIDM Workshop, Miami 1985, pp 107-111.

[Merigot 86a] Merigot A., Garda P., Zavidovique B., Devos F., "Interlayer communications in MIMD pyramidal computers", to appear in Proc. 8th ICPR, Paris 1986.

[Merigot 86b] Merigot A., "Execution d'algorithmes paralleles sur des calculateurs pyramidaux SIMD et MIMD", Internal report.

[Pavlidis 82] Pavlidis T., "Graphics and image processing", Springer Verlag 1982.

[Rosenfeld 84] Rosenfeld A. (ed.), "Multiresolution image processing and analysis", Springer Verlach, 1984.

[Scheafer 86] Scheafer D., "The GAM System", Communication to the Pyramidal Systems for Image Processing Workshop held in Maratea, Italy, May 1986.

[Tanimoto 83a] Tanimoto S.L. "A pyramidal approach to parallel processing", 10th Annual International Symposium on Computer Architecture 83, pp 372-378.

[Tanimoto 83b] Tanimoto S.L., "A boolean matching operator for hierarchical image processing", Proc. CAPAIDM Workshop 83, pp 253-256.

[Tanimoto 83c] Tanimoto S.L., "Algorithms for median filtering on images" in "Computing structures for image processing", Duff MJB Ed., Academic Press 1983.

[Uhr 72] Uhr L., "Layered recognition cones : Netwoks that preprocess, classify, and describe.", IEEE Trans. on Computers, Vol. C-21, No. 7, pp 758-768.

[Uhr 83] Uhr L., "Pyramid multi computer structures, and augmented pyramids", in "Computing structures for image processing", Duff MJB Ed., Academic Press 1983.

COUNTING ON THE GAM PYRAMID

David H. Schaefer and Ping Ho
George Mason University
Fairfax, Virginia 22030/USA

Introduction

The importance of counting in the understanding of images has been underestimated. Problem - distinguishing a man from a dog. A quick solution - count the legs. If the answer is "two", it's a man. Other examples easily come to mind: a hand has five fingers, a pair of scissors has two holes, a teacup has one handle, a bowl none, and the small letter "i" consists of two connected areas.

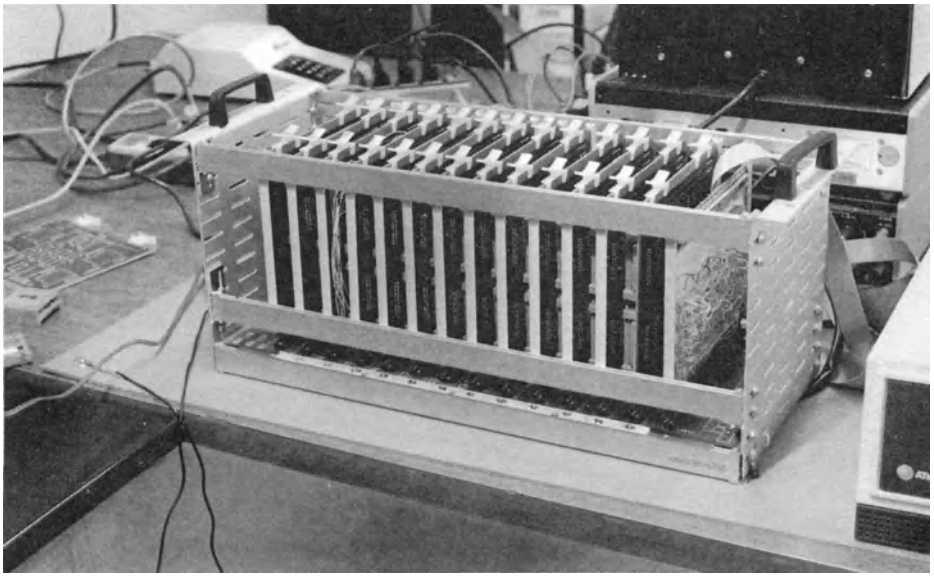


Figure 1 - The GAM Pyramid

At George Mason University we have constructed a five-level pyramid called the "GAM pyramid" [1,2,3] (Figure 1). The initials "GAM" reflect our awareness of the importance of counting as the "A" stands for "adder", in our case a pyramid of full adders that continually presents the count of 1's on the pyramid level above the base of our overall pyramid. The "G" stands for "George Mason University", and the "M" stands

NATO ASI Series, Vol. F 25
Pyramidal Systems for Computer Vision
Edited by V. Cantoni and S. Levialdi
© Springer-Verlag Berlin Heidelberg 1986

for the "Massively Parallel Processor" (MPP) as MPP processing element microcircuits are utilized in the basic pyramid structure. The availability of the MPP chips has allowed us to construct this non-overlapped quad-tree pyramid with student assembly and programming support. The complete structure contains 341 bit serial processing elements, each able to directly communicate with its sibling neighbors to the north, east, west and south, with four children on the level below, and with a parent on the level above. Each of the five levels provides the control unit with the "Global OR" or "Sum-OR" of all data bus values on the level, and in addition (as has already been noted) the level above the base is monitored by the adder pyramid which produces a seven-bit output indicating the number of 1's on its sixty-four inputs.

Connected Region Counting

The primary use of the adder structure is in the determination of the number of connected regions or "blobs" in binary images. This is accomplished through the use of the Euler Characteristic Equation which combines counts of the number of object pixels, edges (any pair of object pixels that constitute part of a horizontal, vertical or diagonal edge) and faces (any two by two area of object pixels). After calculation of the Euler Characteristic, the number of connected regions minus the number of holes in these regions is known. On the GAM structure, the Euler Characteristic number of connected regions on the base of the pyramid can be calculated in less than 30 clock cycles. The time for the adder pyramid operation is counted as one cycle as this hardware will provide an answer within 100 nanoseconds after its input stabilizes.

The determination of the number of connected regions, irrespective of whether they contain holes or not, requires added steps. A frame of 1's must be propagated from the border of the image inward to all objects, until the only zero (or black) areas are the holes. At this point the Euler Characteristic must be used to count these regions of

connected zeros. The number of holes must then be added to the originally computed Euler Characteristic Number to obtain the true number of connected regions. By using the pyramid structure, the propagation of 1's from the border can be accomplished much faster than if the base alone were used for this propagation. Figure 2 shows steps in determining the true number of connected regions.

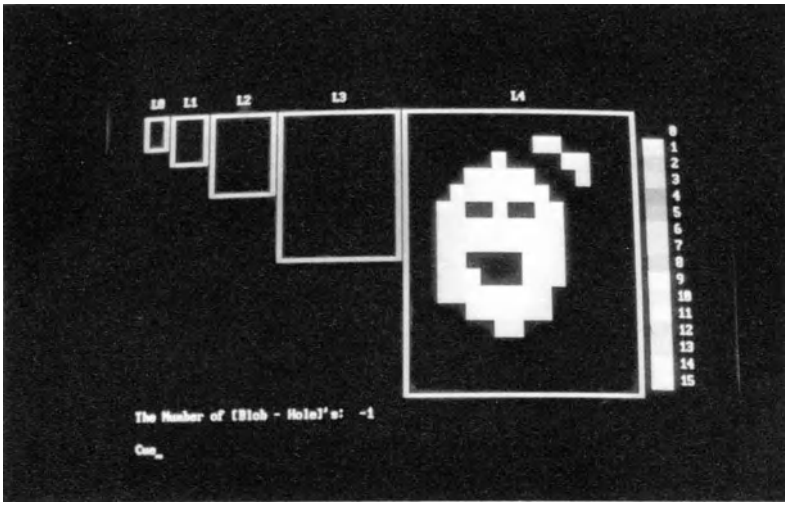


Fig. 2(a) - Determination of number of blobs minus number of holes. The Euler Characteristic Number equals minus one.

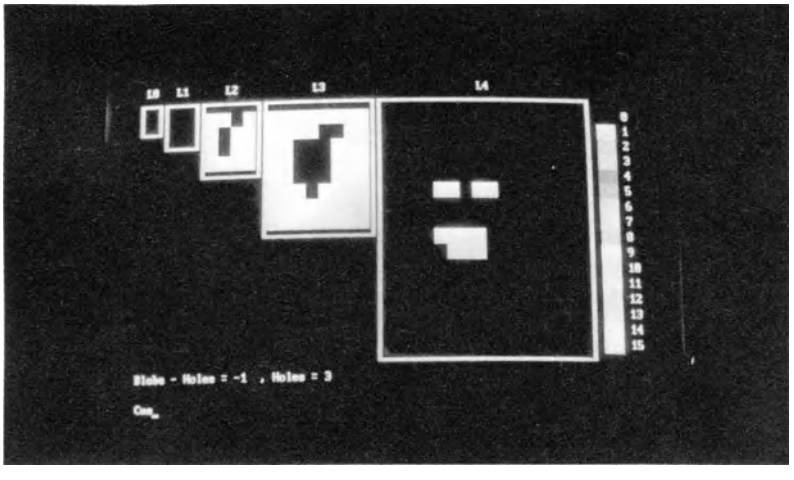


Fig. 2(b) - Detection of holes and determination that two connected regions are present in image. Upper levels show propagation of 1's from edges.

Examples of the Use of Counting Algorithms

The most straightforward use of counting is to determine the area of an object and to use this information for identification. Figure 3 presents an example of where the objects with the largest and smallest areas are separated from other objects in the image.



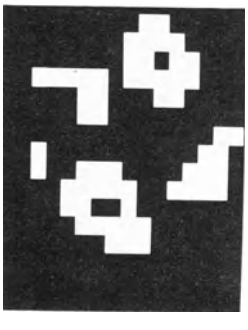
a) Input image



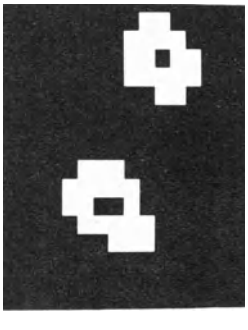
b) Segmented image

Fig 3 - Separation of largest and smallest objects from other objects

A second example of object segmentation that depends on the determination of the Euler Characteristic is shown in Figure 4. Here objects with holes are separated from objects without holes. The algorithm consists of first segmenting the image to find each object, and then utilizing the Euler Characteristic to identify those with holes.



a) Input image



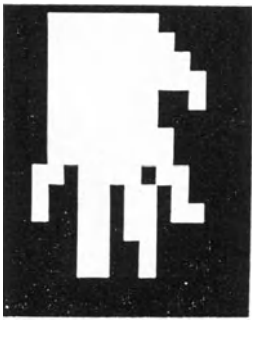
b) Separated image

Fig. 4 - Separation of objects with holes

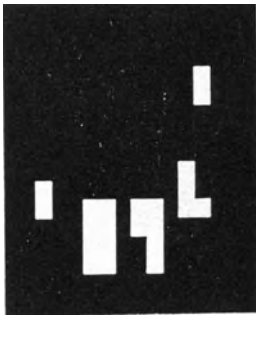
One of the first demonstrations on the GAM system was the counting of outstretched fingers on a hand placed in the field of view of a television camera. Figure 5 shows various steps in this procedure. The algorithm identifies the palm area of the hand by its large area, removes this from the image, and then utilizes the Euler number to count the now isolated fingers.



(a) Hand of author being observed by television camera



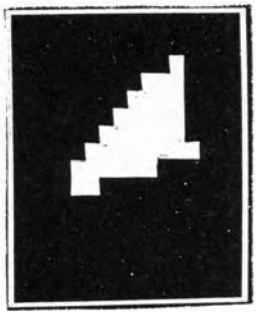
(b) Hand on base of pyramid



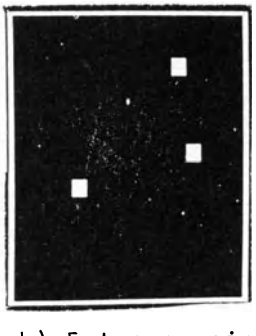
(c) Fingers segemented and counted

Fig. 5

The identification of triangles is carried out by first determining the extreme points of a sensed object. If there are either more or less than three of these extreme points, then the object is not a triangle. If three extreme points are found, there still remains a question as to whether the object is a triangle. Therefore, a triangle is constructed based on the location of the three extreme points of the object by having each processing element calculate whether it is inside or outside of this mathematical triangle. If the unkown object and the constructed triangle have a sufficient percent of pixels in common, then the object is judged to be triangular. Counting enters into this algorithm twice - once to determine the number of extreme points and, secondly, to determine the amount of overlap of the unknown object and the constructed triangle.



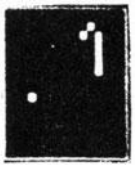
a) Sensed object



b) Extreme points of sensed object



c) Constructed triangle



d) Pixels not in both constructed triangle and sensed object

Fig. 6 - Steps in Triangle Identification

The method used to distinguish between an "L" shaped object and a "T" shaped one also involves counting. After identification and segmentation of the arms of the unknown object, the intersection of the two arms is removed from the image. If the object is "L" shaped, two blobs will remain. If the object is "T" shaped, three blobs will remain.
A final example is to identify a dumbbell shaped object. Such an object will break into two parts at some upper level of an AND pyramid.

Concluding Remarks

The importance of counting hardware and software has been demonstrated using the GAM pyramid to perform image recognition tasks. The ability to rapidly count will be a prime requirement in future image understanding hardware.

Acknowledgments

The authors wish to acknowledge the contributions of John Boyd, Ceser Vallejos, Man Bun (David) Chu and Zahi Abuhandeh to the GAM system. They also wish to acknowledge the contributions of Maxine Schaefer in the preparation of this paper for publication.

References

1. Schaefer, D.H., Wilcox, G.C. and Harris, V.J. "A Pyramid of MPP Processing Elements - Experience and Plans," Proceedings of the 18th Annual Hawaii International Conference on System Science, Jan. 1985.
2. Schaefer, D.H. and Wilcox, G.C. "The MPP Pyramid Computer" Proceedings of the 1985 IEEE International Conference on Systems, Man and Cybernetics, Tucson, Arizona, Nov. 1985.
3. Schaefer, D.H. "Pyramid Architectures". In M.J.B. Duff (Ed.): Intermediate Level Image Processing. London, Academic Press (in press).
4. Schaefer, D.H., Ho, P., Boyd, J., Vallejos, C. "The GAM Pyramid". In L. Uhr (Ed.): Multi-Computer Pyramids for Perceptual Recognition. New York, Academic Press (in press).

A Pipelined Pyramid Machine

P. J. Burt, C. H. Anderson, J. O. Sinniger, G. van der Wal
RCA David Sarnoff Research Center

1. Introduction

Interest in pyramid processing has grown rapidly over the past few years. This is due in large part to the emergence of hierarchical models for information processing in human vision, and to the development of fast algorithms for image filtering and pattern matching. But it is also due to the fact that even powerful pyramid algorithms are built from simple neighborhood operations, operations that are well suited for implementation in special purpose computing hardware.

The pyramid has been proposed both for multiresolution signal processing and for semantic information processing. Justification for the former begins with the observation that salient pattern information occurs at many scales within an image. In order to match operator to pattern scale, it is expedient to represent the image as a "resolution pyramid", a set of band pass or low pass copies of the image that are appropriately subsampled. This provides a framework for highly efficient filtering, feature extraction, and coarse - fine search, as well as a framework for organizing pattern information over a wide range of scales (Rosenfeld, 1984). Numerous analysis procedures have been based on the multiresolution pyramid, including pattern recognition and motion analysis techniques reported at the Maratea workshop (Dengler, 1986; Neven, et. al. 1986; Dyer, 1986; Zimmer,1986).

Justification for semantic information processing begins with a conjecture about how an image is transformed as it flows through the human visual system. Initially the image is repre-

NATO ASI Series, Vol. F 25
Pyramidal Systems for Computer Vision
Edited by V. Cantoni and S. Levialdi
© Springer-Verlag Berlin Heidelberg 1986

sented at high resolution in terms of simple pattern elements. These elements are then combined into more and more complex structures at successive stages of processing, until elements are constructed which carry semantic information about individual objects in a scene. It is reasoned that as pattern information becomes more complex, position information becomes less precise: there is a gradual shift from encoding "where" to encoding "what". Thus successive stages of processing fit naturally into a pyramid structure, with semantic content increasing and resolution decreasing from level to level (eg. Uhr, 1983).

Several special purpose hardware architectures have been proposed for pyramid processing, some of which are now being constructed. For the most part these are mesh - connected machines, consisting of arrays of simple bit serial processors, one processor for each pyramid node (Cantoni, et. al, 1985; Merigot, et. al, 1985; Scheafer, 1986). Each processor is connected to its neighbors in the same pyramid level, and to its father and children processors in neighboring levels. All processors within a given level perform operations in parallel, SIMD mode, while different levels may perform different operations, MIMD mode.

An alternative architecture for pyramid processing is based on a pipeline of processors. Rather than use a large number of simple processors, just one or a small number of high performance processors are arranged in sequence. Data are stored in memory separately from the processors. Computations are performed by removing data sequentially from memory, passing the data stream through the processor pipeline, then returning the results to memory. Data may be recirculated for further computations.

Here we describe a pipelined pyramid machine that has been built and tested over the past two years. The machine runs at video rates and is designed specifically for multiresolution signal processing. As such it is well suited for a wide variety of early vision tasks, including motion and stereo analysis, and hierarchical pattern recognition.

2. Basic Pyramid Operations

Four types of operations are used in pyramid - based signal processing: filter convolutions, point operations, image - image operations and local integration. Filtering is required for pyramid generation and feature enhancement. Integration is used in computing various local property measures. All of these operations are generally applied uniformly over entire pyramid levels, or within specified subregions, so they can be performed effectively by special array processing hardware.

Gaussian Pyramid

Only operations for pyramid construction need to be described here. The same operation can serve local integration, while point and image - image operations are standard.

To begin we consider a Gaussian, or low pass, pyramid. This is generated through a sequence of filter and decimate steps. Let I be an original image, which for simplicity, we assume measures 2^N by 2^N pixels. Let G_0, G_1, ..., G_N be levels of the Gaussian pyramid constructed from I. Then G_0 is just the original image, while successive pyramid levels are computed recursively through repeated filter and subsample steps:

$$G_0(i,j) = I(i,j) \quad \text{for } 0 \le i,j < 2^N$$

and

$$G_l(i,j) = \sum_{m,n=-2}^{2} w(m,n)\, G_{l-1}(2i+m, 2j+n) \quad \text{for } 0 \le i,j < 2^{N-1} \text{ and } 0 < l \le N.$$

Here w, the generating kernel, acts as a low pass filter. It is assumed to measure 5 by 5 because that has proven adequate for many processing and analysis tasks, and because this size has been implemented in our hardware. Larger or smaller kernels may be used. The weights of the kernels are selected in accord with certain constraints, including a stipulation that w be separable so that it may be implemented efficiently as a cascade of two

one-dimensional filters (Burt, 1981).

It is useful to cast this definition of the Gaussian pyramid in terms of separate filter and decimate steps. Let $\hat{G}_l$ be the result of filtering G_{l-1} without decimation. Then construction is a standard convolution, $\hat{G}_l = w * G_{l-1}$, for l=1 to N, followed by decimation,

$$G_l(i,j) = \hat{G}_l(2i,2j) \quad \text{for} \quad 0<i,j\leq 2^{N-l}.$$

FSD Laplacian

The Laplacian, or band pass, pyramid is constructed as the difference between successive levels of the Gaussian. Since levels differ in sample density the subtraction must be performed before decimation or after missing samples have been re-interpolated. We define two types of Laplacian pyramids which differ in this aspect of construction. For the first, "filter - subtract - decimate" or FSD, the difference operation is performed before decimation. Let L_0, L_1, ..., L_{N-1} be levels of the Laplacian pyramid. Then we simply define

$$L_l = G_l - \hat{G}_{l+1}.$$

The name refers to the fact that Laplacian and Gaussian pyramids can be constructed simultaneously in hardware. Each Gaussian level is filtered, then subtracted from its unfiltered version to form the corresponding Laplacian, before it is decimated to form the next Gaussian level (Anderson, 1984).

RE Laplacian

In the second definition of the Laplacian, called "reduce - expand" or RE, each level is computed from the corresponding Gaussian level and the next level after decimation. In order to construct a difference at level l, G_{l+1} must be "expanded" to the size of G_l. This is done by interpolating new samples between

the given samples. We define the interpolation process as follows. Let $G_{l,k}$ be the result of expanding G_l k times. Then

$$G_{l,0} = G_l,$$

and

$$G_{l,k}(i,j) = 4 \sum_{m,n=-2}^{2} w'(m,n)\, G_{l,k-1}\left(\frac{i+m}{2}, \frac{j+n}{2}\right).$$

Here only terms for which i+m and j+n are even are used in the sum. The factor of 4 compensates for the fact that only a quarter of the terms in the kernel are used in a given sum. The interpolating kernel, w', is chosen subject to the same constraints as the generating kernel, w, used in Gaussian pyramid construction, although the kernel weights need not be the same.

Again this definition includes two distinct steps, the insertion of new sample points and a filtering operation. It is convenient to specify these separately in the definition. Accordingly, let $\hat{G}_{l,1}$ represent the result of inserting new sample points between the given samples of G_l. The new samples are given the value 0:

$$\hat{G}_{l,1}(i,j) = \begin{cases} G_l(i/2, j/2) & \text{for i and j even} \\ 0 & \text{otherwise.} \end{cases}$$

Thus $\hat{G}_{l,1}$ is just G_l "padded" with zeros. Next $G_{l,1}$ is formed by convolving $\hat{G}_{l,1}$ with kernel w':

$$G_{l,1} = 4\, w' * \hat{G}_{l,1}$$

Again the factor of 4 compensates for the fact that 3 of 4 samples in the padded array, $\hat{G}_{l,1}$, are zeros.

Now the RE Laplacian is defined

$$L_l(i,j) = G_l(i,j) - G_{l+1,1}(i,j).$$

The term "reduce - expand" refers to the fact that each Laplacian level is formed as the difference between the corresponding Gaussian level and an image obtained by reducing that image (filter and decimated) then re-expanding (pad with zeros and filter). The RE Laplacian has an advantage for some applications because the original image can be recovered exactly from its Laplacian pyramid. This is useful, for example, in image compression (see Burt and Adelson, 1983). The FSD form, on the other hand, can be computed with somewhat simpler hardware, and is suited for analysis applications where image reconstruction is not an objective. Both forms can be generated in our pyramid.

Local Integration

Constructing the Gaussian pyramid has the effect of convolving the original image with a set of Gaussian - like weighting functions. We refer to these as "equivalent weighting functions," W_1, W_2, ..., W_N. G_1 is just W_1 convolved with the original image followed by decimation. Alternatively, these weighting functions may be viewed as integration windows: each sample of G_1 represents a weighed average of pixels of the original image within a window W_1 centered at a corresponding image position. Thus pyramid construction serves to integrate image information within local windows. The computation is very efficient, and generates results for many window sizes simultaneously. This is important in analysis when the appropriate window size is not known a priori, or in coarse - fine search where analysis begins with large windows and proceeds to small.

An Algebra

It is convenient to introduce a shorthand notation, or

algebra, for describing computations within a pyramid structure. Let $G_l[I]$ be the lth level a Gaussian pyramid for initial image I, and let $L_l[I]$ be the lth level of a Laplacian pyramid constructed for this image. Thus square brackets indicate that pyramid operations are performed on the image specified within the brackets. Brackets may be nested to represent a sequence of operations. For example $G_2[L_0[I]]$ indicates that a Laplacian pyramid is first constructed for image I, then Gaussian level G_2 is constructed using L_0 as the starting image. Each sample of the resulting image represents the average value of L_0 computed within a Gaussian shaped window specified by the equivalent weighting function W_2. In this example the first pyramid operation may be interpreted as a band pass filter, while the second serves as an integration process.

Point operation and image - image operations are represented in standard algebraic notation with image array names as variables. Thus $G_k[\{L_l[I]/G_{l+1,1}[I]\}^2]$ indicates that Laplacian level L_l and Gaussian level G_{l+1} are first formed for image I. Then G_{l+1} is expanded once to match the sample density of L_l, and each sample of the Laplacian is divided by the corresponding Gaussian sample. The result is then squared on a pixel by pixel basis, and integrated locally through the second Gaussian construction process.

3. A Pyramid Machine

We have constructed a machine to perform pyramid image processing at video rates. This combines a pyramid processing module with a host micro computer.

Pyramid Processing Module

Five computational units are required for pyramid construction: a filter, a decimator, an expander (to insert zeros), and ALU (to form the sum and difference between images), and several image frame stores.

An arrangement of these components for constructing a Gaussian pyramid is shown in Figure 1a. The process begins with the original image in one frame store. This image, G_0, is read from the store in raster scan format, passed through a filter unit, decimated, and returned to a second frame store as G_1. Once G_1 is complete, it is read from the store, filtered, subsampled and returned as G_2, and so on.

The filter unit is designed to perform a 5 by 5 convolution. Thus to compute any output sample, five successive rows of the input image must be available to the filter unit. One row is provided directly from the frame store, as shown, while the four previous rows are circulated through FIFO (first in, first out) line stores. The output of the filter, $\hat{G}_{l+1}$, is passed through a decimator which discards every other row and column of data.

The FSD Laplacian pyramid is constructed as shown in Figure 1b, by subtracting samples of the filtered image, $\hat{G}_{l+1}$, from samples of the image being filtered, G_l. A given output sample is subtracted from the input sample which is in the center of the 5 by 5 array, and hence in the center line store at the input of the filter, as shown.

Since the decimator discards three out of every four samples in the filtered image, considerable improvement in speed and efficiency can be realized simply by not computing these samples in the filter unit. This is possible when we are only interested in generating Gaussian pyramid levels. Intermediate filtered samples are required, however, when computing the FSD Laplacian pyramid.

Two passes through the filter are required to construct the RE Laplacian pyramid. Suppose Gaussian levels up to G_l and Laplacian levels up to L_{l-1} have been constructed and are saved in the frame stores. G_l is read from memory, filtered, decimated, and returned to memory as G_{l+1}. G_{l+1} is then expanded and subtracted from G_l to form L_l, as shown in Figure 1c. To expand G_{l+1}, it is first passed through an expanding unit where zeros

are inserted between the given samples. The resulting image, $\hat{G}_{l+1,1}$, is then passed through the filter to complete the interpolation process. As samples of $G_{l+1,1}$ are generated they are subtracted from corresponding samples of G_l, which are simultaneously read from a second frame store.

Although construction of the RE Laplacian requires two filter operation while the FSD requires only one, the RE can actually be computed somewhat faster if the filter is designed to take advantage of subsampling of G_{l+1}.

The original image can be reconstructed from its RE Laplacian with the same filter structure shown in Figure 1c. Suppose level L_0, ..., L_{N-1} of the Laplacian, and the low pass residual G_N are available in the frame stores. G_N is read from memory, expanded and added to L_{N-1} to form G_{N-1}. This is expanded and added to L_{N-2} to form G_{N-2}, and so on until the original image, G_0, is recovered.

A full module for pyramid construction combines the various circuits outlined in Figure 1. Multiple modules of this type could be cascaded, or run in parallel to increase the overall processing speed.

Special attention must be given to boundary points during pyramid construction. In effect a pyramid level should be extended by two sample positions along each boundary before it is passed through the 5 x 5 filter. Various types of extrapolation can be used, including simply inserting a constant such as zero, copying the boundary value, reflection at the boundary or linear extrapolation. Mechanisms for performing this extension (in our case inserting a constant or copying the boundary value) are included in the filter unit itself.

System Design

We have constructed a prototype machine vision system based on the pyramid module outlined above. Further details of the

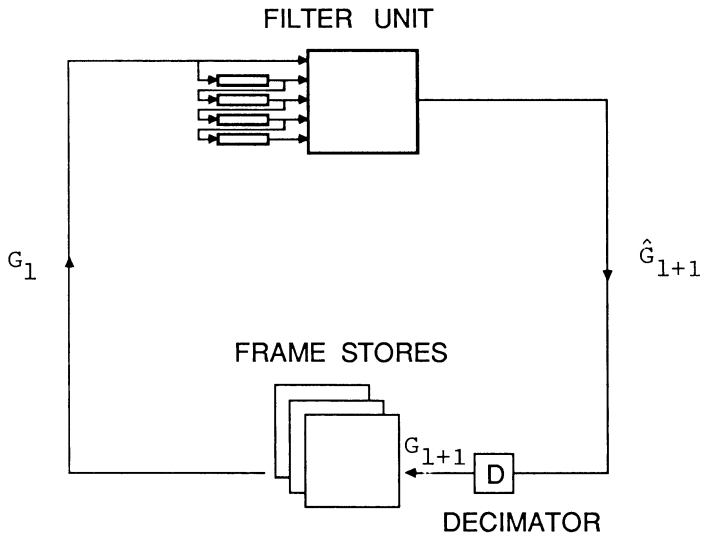


Figure 1a. Gaussian pyramid construction.

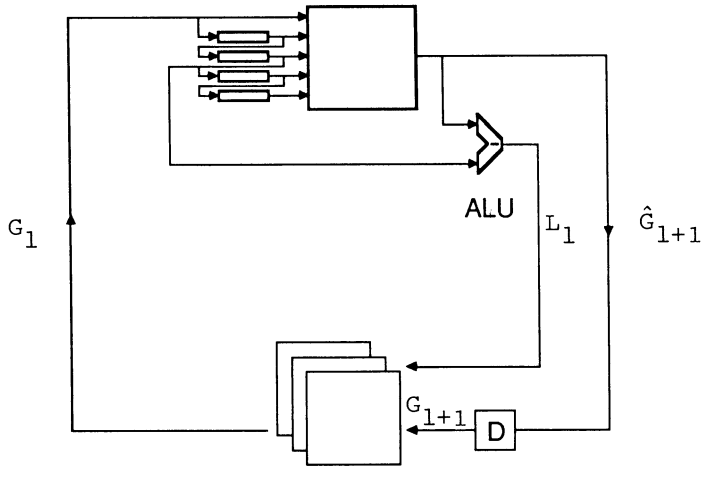


Figure 1b. FSD Laplacian construction

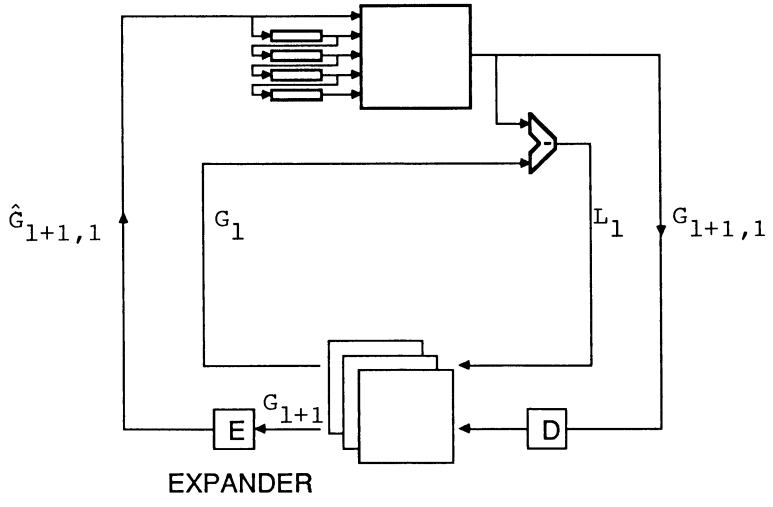


Figure 1c. RE Laplacian construction.

system are given in van der Wal and Sinniger (1985), while a related system for video processing is described by Arbeiter and Bessler (1986).

A general outline of the machine vision system is given in Figure 2. The pyramid boards contain the filter unit, delay lines, and an ALU for level subtraction and addition. In addition a look-up table has been included at the output of the filter so that point operation, such as square or absolute value, can be performed simultaneously with pyramid construction.

The decimate and expand - with - zeros functions have been built into the memory units. Samples may be dropped as data is written to memory, while zeros may be inserted as data is read from memory. Dual port memory is required so that image data can be accessed by the host computer as well as the filter unit.

The ALU unit shown in Figure 2 is in addition to the unit on the pyramid boards. It is used for array processing other than

pyramid construction, such as convolution with arbitrary kernels.

Finally, a host, in our case a 68000 based micro computer, controls operations in the pyramid, ALU and memory devises. The host also performs any other computations required in image analysis, such as pattern classification. Machine vision application are implemented with software written on the host.

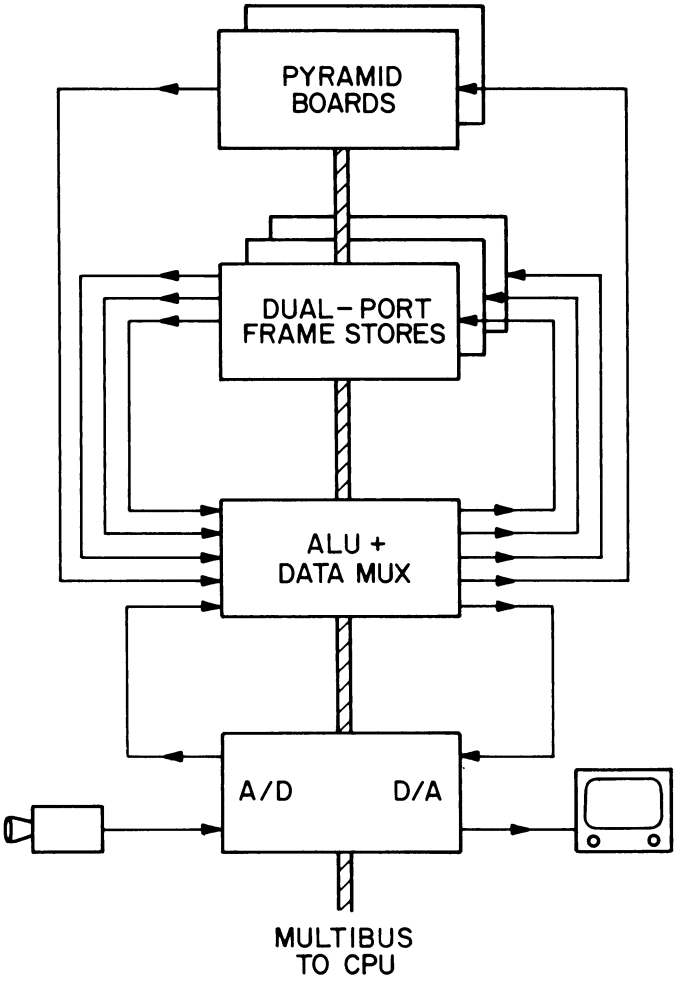


Figure 2. Pyramid machine.

3. An Example

To illustrate use of the pyramid machine we show how it may be included in a "smart camera" for automated surveillance. A camera - processor device is required that can monitor a scene, such as a parking lot, and alert a guard when critical events occur, such as the movements of cars or people within the lot. It is essential that the system be sufficiently "clever" to distinguish between events of interest and other activities in the scene, such as the movement of animals, trees and clouds. But it is equally essential that all analysis be performed on a small inexpensive system.

A surveillance camera typically generates 30 image frames per second, each measuring perhaps 256 by 256 or 512 by 512 pixels. This data rate vastly exceeds the computation capabilities of even large general purpose computers. We may view the pyramid as an image preprocessor that serves to extract essential image features from raw image data, and reduce the data rate to a level that can be analyzed even by a micro computer.

A number of features could be computed within the pyramid to detect and discriminate between the various types of events that occur in a parking lot scene. Here we illustrate by computing local estimates of space - time spectral energy. Scene changes due to passing clouds can then be identified because energy is concentrated at low spatial and temporal frequencies. Leaves blowing in a breeze, on the other hand, have a predominance of spectral energy at high spatial and temporal frequencies. Events of interest, such as the movements of people and cars, are characterized by energy within intermediate frequencies(Anderson, et. al., 1986).

Figure 3 outlines a four stage procedure for computing "change energy" within the pyramid machine. Image frames from the camera are digitized, and difference images are formed by subtracting each frame from its predecessor. Frame difference acts as a high pass temporal filter. Each difference image is

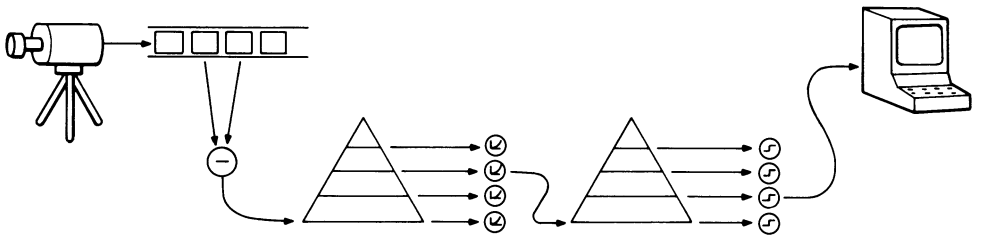


Figure 3. Procedure for computing "change energy".

then decomposed into a set of spatial band pass components through construction of a Laplacian pyramid. The band which best characterizes changes of interest is selected for further processing. Its values are squared (using the look-up table on the pyramid boards) then integrated locally. Integration is implemented simply by building a low pass, Gaussian, pyramid from the squared image. Each sample in this "change energy pyramid" represents energy within the selected frequency band integrated within a Gaussian - like window, with the window size doubling from level to level of the pyramid. Pyramids generated at successive stages of this computation are shown in Figure 4.

The microprocessor accesses data in the change pyramid, and detects and tracks objects moving in the scene through a coarse - fine search, or "foveation", strategy. In our system the microprocessor is capable of examining roughly a 16 by 16 subarray of data each frame time. The search begins at pyramid level G_4 which contains just 16 by 16 samples. Although much reduced in size and resolution, this data represents image change wherever it may occur within the original image. If a change is detected at G_4, the microprocessor examines a 16 by 16 subarray of G_3 centered on that change. This subarray represents only one

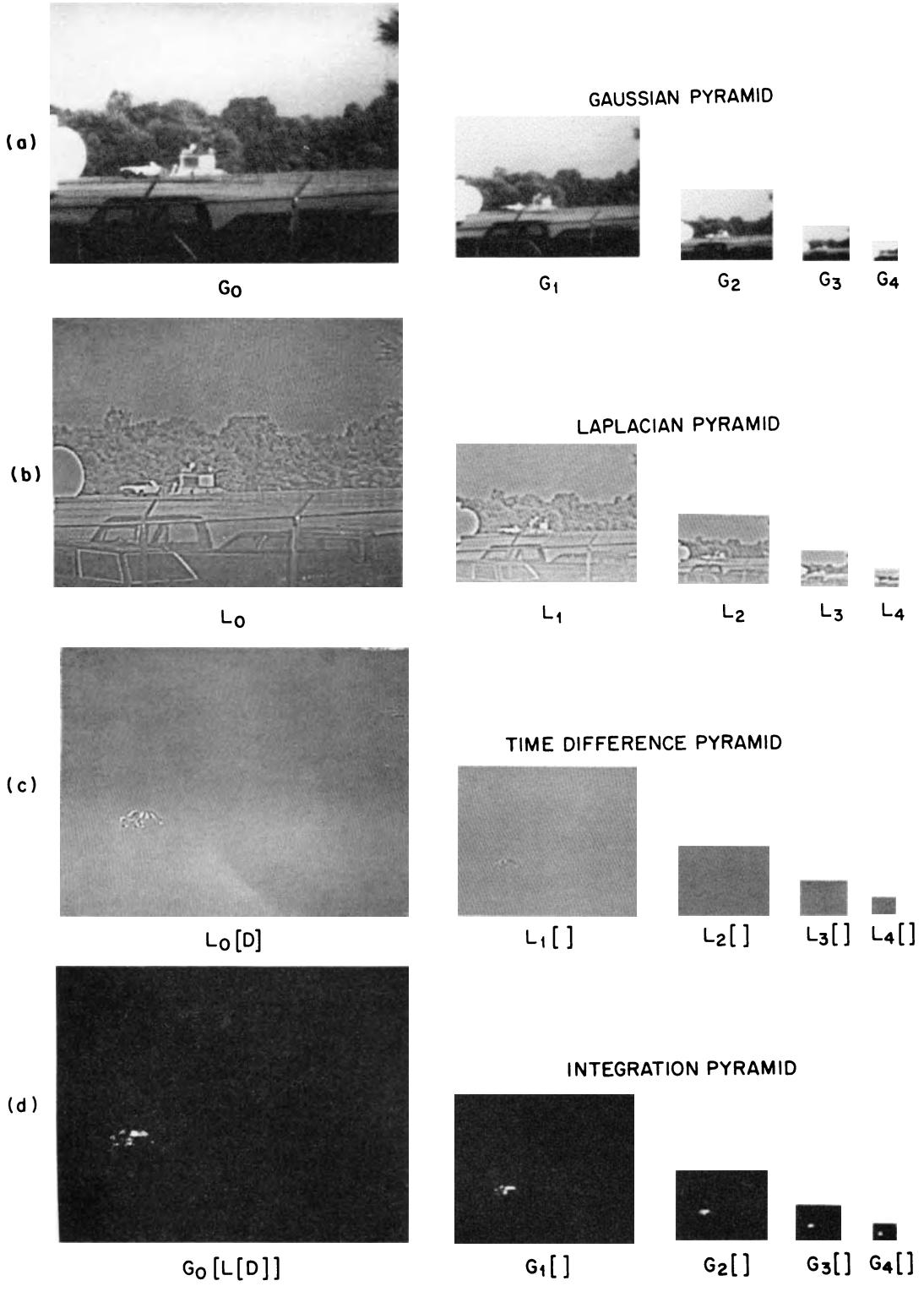


Figure 4. Four stages in computing change energy.

fourth of the entire image domain, but it should contain the region of interest. A change detected within the new subarray directs the microprocessor to examine a 16 by 16 subarray of G_2. In this way the system can rapidly home in on small objects anywhere within the field of view even though the microprocessor examines only a small fraction of samples required to represent the image.

The homing process is shown in Figure 5, where small markers are superimposed on the camera images to indicate the region being examined by the microprocessor in successive frame times. The sequence shows the system monitoring the entire field of view then homing in on a moving car. Again the amount of data examined at each stage is constant, a 16 by 16 array, but this represents change energy at successively higher resolution.

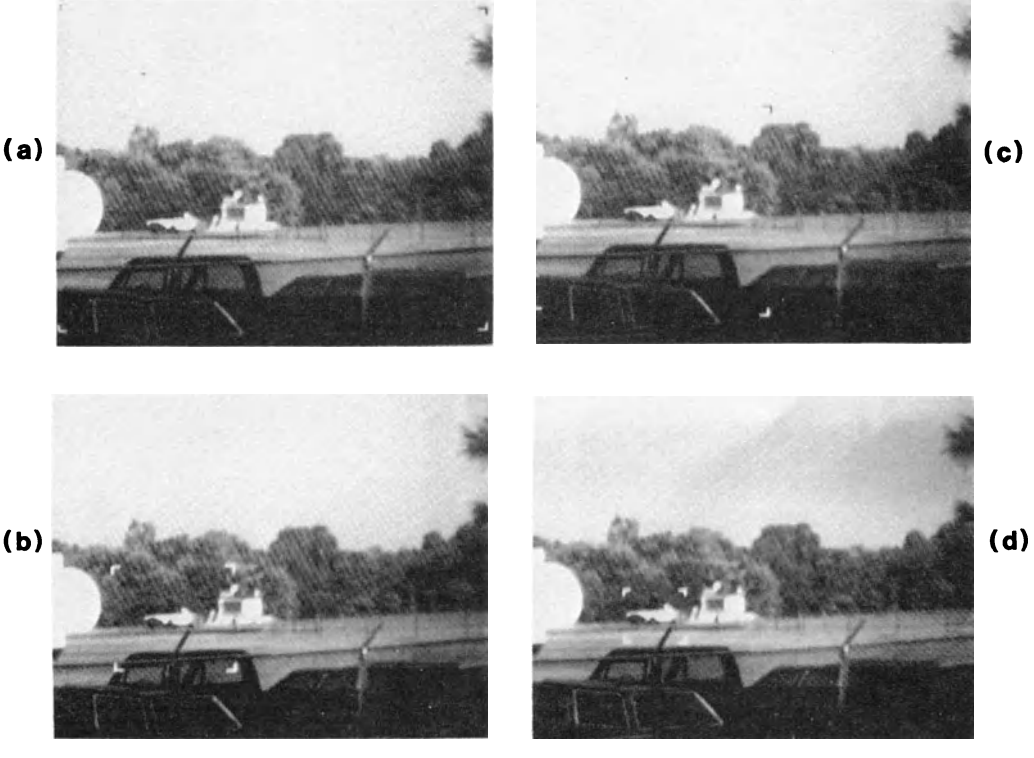


Figure 5. Homing process over four frames

Three basic functions of pyramid processing are illustrated in the surveillance example. First, through decomposition of the source image into a pyramid structure, analysis operations can be matched effectively to the scale at which salient information appears in the image. Second, through repeated applications of pyramid and point operations the system can rapidly generate arrays of image features, such as change energy. Third, further analysis by the microprocessor is restricted to small subregions of data that can be directed to the region of interest though course - fine search, or foveation. The particular steps outlined here are intended just as an example. Many other analysis tasks can be implemented within the same general framework. Taken together, these pyramid techniques typically provide a speed up of several orders of magnitude over conventional analysis techniques.

5. Comparison to Mesh - Connected Machines

Here we have described a pipelined approach to pyramid - based image processing. It is appropriate to ask how such an architecture may compare with the mesh - connected multi - processor architectures under study by several other groups. In particular, for given hardware complexity, or cost, which design provides the greater computing power?

We will compare the two designs by estimating how long each would take to perform a typical sequence of processing steps. This sequence begins with the construction of a six level pyramid from an initial 256 by 256 image. We assume a 5 by 5 filter is used in pyramid construction and that computations are carried out to 16 bits, then the output is truncated to 8 bits. All further processing is restricted to 16 by 16 or 32 by 32 analysis windows within the pyramid levels. Windows are analyzed sequentially in a coarse to fine fashion. A number of different feature selective operations are performed in each window. This scenario is typical of procedures we have found useful in image analysis.

The pipeline machine is assumed to have the following characteristics. It can construct an entire pyramid for a 256 by 256 image in 30 msec.. This is true for our present machine, although future designs will be substantially faster. Processing time is proportional to image size, so that operations on reduced resolution pyramid levels, or reduced size analysis windows, can be carried out at substantially higher rates. "Proportional time" is particularly important property of pipelined processing. Finally we believe the pyramid filter unit can be integrated on a single VLSI chip.

Next, the following assumptions are made for the mesh machine. A separate, bit serial processor is allocated to each pyramid node. The base level of the pyramid measures 256 by 256, so matches the size of the input image. Thus the mesh machine has a total of 85K processors. Processors are connected to their nearest neighbors in the same pyramid level, and to father and children processors in neighboring levels. Processors within each level work together in SIMD mode, while different levels may perform different operation simultaneously, in MIMD mode. The clock rate is 10MHz. Finally, we assume that 16 such processors can be integrated on a single VLSI chip. These numbers are typical of current designs.

The mesh machine will require roughly 500 clock cycles to construct each pyramid level. This is the time to communicate data for the 5 by 5 convolution neighborhood one bit at a time, followed by several bit serial multiplies and adds. The filter process must be repeated five times to construct the six level pyramid. It follows that the mesh machine can complete the image pyramid in 250 micro seconds, or in about one hundredth the time required by the pipeline machine.

This apparent speed advantage of the mesh design changes dramatically when we state processing power in terms of cost. At 16 processors per chip the mesh machine requires roughly 5000 chips, while the pipe processor requires just one. It is possible to run multiple pipe processors in parallel, each

constructing a pyramid for a subregion of the image, or in sequence, each assigned to a given pyramid level, and thereby obtained a nearly proportional increase in speed. A system of 100 pipelined processors could build a pyramid in the same time as the full mesh machine. For equal compute power for pyramid construction the mesh design would require 50 times more chips than the multiple processor pipeline design.

But there appears to be a more serious problem with the mesh design. This arises when processing is restricted to low resolution pyramid levels, or to analysis windows within high resolution pyramid levels. Processing time with an SIMD array and a processor assigned to each node is independent of image size. Unlike the pipeline design, the mesh requires just as much time to process a 16 by 16 window as the 256 by 256 original image. It is possible to perform window operations in a mesh machine by turning off processors outside the window, but this only means machine efficiency is drastically reduced. In our example the preponderance of processing is performed within 16 by 16 or 32 by 32 windows. For such operations a single pipeline processor can out - perform a full mesh - connected pyramid machine.

Thus we find that the pipe design appears to have two advantages over the mesh design, at least for the the types of processing we find useful in image analysis. It provides greater processing power per unit cost, and it is more efficient (and flexible) when processing within windows. But we do not wish to conclude that a simple pipeline is preferable to parallel processing. Rather we believe that a system consisting of a number of pipelined processors running in parallel is particularly well suited for image analysis. When processing large images each processor would be assigned to a different image subregion, and all processors would perform the same operations in SIMD mode. But when processing small images or windows, each processor would be given a different analysis task to be performed on the same data array in MIMD mode.

References

Anderson, C. H., "An alternative to the Burt pyramid algorithm," RCA Correspondence, 1984.

Anderson, C. H., P. J. Burt, and G. S. van der Wal, "Change detection and tracking using pyramid transform techniques," Proc. SPIE Conference on Intelligent Robots and Computer Vision, Boston, pp. 72-78, 1985.

Arbeiter, J. H., and R. F. Bessler, "A two-dimensional real-time video pyramid processor", RCA Review, vol. 47, pp. 3-31, 1986.

Burt, P. J., "Fast filter transforms for image processing", Comp. Graphics and Image Processing, 16, pp. 20-51, 1981.

Burt, P. J. and E. H. Adelson, "The Laplacian pyramid as a compact image code", IEEE Trans. Comm., COM-31, pp. 532-540, 1983.

Cantoni, V., L. Carrioli, O. Catalano, L. Cinque, V. Di Gesu, M. Ferretti, G. Gerardi, S. Levialdi, R. Lombardi, A. Machi, R. Stefanelli, "The PAPIA image analysis system," SPIE International Symposium on Image Processing, Cannes, 1985.

Dengler, J., "Local motion estimation with the dynamic pyramid", in this volume, 1986.

Merigot, A., B. Zavidovique and F. Devos, "SPHINX, a pyramidal approach to parallel image processing," Proc. IEEE Workshop on Computer Architectures for Pattern Analysis and Image Data Base Management, Miami Beach, FL, pp. 107-111, 1985.

Nevek, C. F., C. R. Dyer, and R. T. Chin, "Object recognition using multiresolution models", Comp. Vision, Graphics and Image Proc., pp. 52-65, 1986.

Rosenfeld, A., Ed., Multiresolution Image Processing and Analysis, Springer,
Berlin, 1984.

Scheafer, D., this volume, 1986.

van der Wal, G. S., and J. O. Sinniger, "Real time pyramid transform architecture," Proc. SPIE Conference on Intelligent Robots and Computer Vision, Boston, pp. - - -, September, 1985.

Uhr, L., "Pyramid multicomputer structures and augmented pyramids," in M. J. B. Duff, Ed., Computing Structures for Image Processing, Academic Press, London, pp. 95-112, 1983.

Zimmer, H.-G., "Vectorial features in pyramid image processing," in this volume, 1986.

THE PAPIA CONTROLLER: HARDWARE IMPLEMENTATION

Gaetano Gerardi
Istituto di Fisica dell'Universita'- Via Archirafi 36-Palermo
IFCAI/CNR- Via M. Stabile 172 - Palermo

ABSTRACT

After a general introduction about parallel machine architectures and possible implementation of control units, extensive informations are given about the hardware implementation of the control unit realized for the PAPIA machine in the Palermo IFCAI/CNR electronic laboratories.

Key words: Parallel computer- microprogrammed architectures- Image analysis

INTRODUCTION

At present, computer science, in the effort of producing computers with increasing speed, advantaged by the decreasing cost of electronic components, is engaged to make parallel computers with special or general purpose characteristics. In general such a computer consists of several processing elements (PE) that can operate simultaneously. A problem to be solved, is broken into a number of subproblems, each of which is solved on one of the processors. The way to solve problems is strongly driven by the architecture choosed. The converse is also true.

A number of multipurpose parallel architectures have been proposed in order to have a parallel machine designed for solving a broad range of applications.

1. PARALLEL MACHINE CLASSIFICATION

Parallel machines are usually classified in two main categories:

-Single Instruction Multiple Data stream (SIMD)
-Multiple Instruction Multiple Data stream (MIMD)

In between there is a third category:

-Multiple Single Instruction Multiple Data stream (MSIMD)

A SIMD computer consists of a number of processing elements operating under control of an intruction stream issued by only one CONTROL UNIT. Each PE has a small private memory for storing data and operates syncronously i.e. during a given clock cycle a number of enabled PEs executes the same instruction. PEs exchange data using an interconnection network or a common memory.

NATO ASI Series, Vol. F 25
Pyramidal Systems for Computer Vision
Edited by V. Cantoni and S. Levialdi
© Springer-Verlag Berlin Heidelberg 1986

Several different configurations have been proposed for the interconnection network: linear, mesh, tree, etc. Similarly, many models of the shared memory approach exist.

In a MIMD computer, processors have indipendent instruction counters and operate asyncronously. As with the SIMD category different configurations have been proposed.

The MSIMD computer consists of a collection of SIMD each having his own control Unit [1].

A pyramidal architecture machine is a MSIMD. In fact we can cut it horinzontally in a number of parts and we can dedicate a control unit to each one; one is still a pyramid the other parts are flat arrays. Each section, in dependence of the work organization of the dedicated control units, can be operated syncronously or asyncronously.

Later on we will concentrate our attention to describe our choice for the control unit of the PAPIA (Pyramidal Architecture for Parallel Image Processing) machine. Arguments will be introduced in a general way.

2. THE PAPIA PROCESSING ELEMENT

The PAPIA PE is a one bit serial processor with private 256 bits memory and 1 Mhz processing clock (CLK1).

Exstensive description of ALU operations, Boolean Registers and elementary functions are given elsewhere. [2], [3], [4].

PEs are quad-tree interconnected: one father, four brother, four sons.

The chip package implementation is so that in a single chip coexist five PEs, internally connected as one father and four sons. Connections at external brothers and sons are provided by related pins in the chip. Due this implementation the number of possible pyramid layers is even. Each son in a package has an input/output register (I/O) pair connected. Each pair has external connection possibility to two other sons on the same layer (brothers) and line. They work so that each raw is like a serial shift register controlled by an external clock (CLK2). It runs faster and asyncronously respect to the processing clock (CLK1). All I/O registers on a layer work as a First-in First-out (FIFO) memory. The parallelism and the depth of the FIFO, is equal to the number of PE in a raw or column (N). (Square base for the pyramid is choosed)

The I/O registers are used to input or output images by bit-plane.

INPUT: a bit-plane is completly trasfered from an external memory, in column mode, after N CLK2 steps. Only one store instruction is enough to transfer the bit-plane in memory; each bit is stored in the addressed bit of the enabled PE memory on the layer.

IN OUTPUT operations, only one instruction is used to load the I/O registers on a layer from the accumulator of enabled PE. N CLK2 steps transfer the bit-plane to the external memory by column.

At the base of each pyramid section external edge registers are provided in order to implement image scroll functions.

An Active Memory Unit is dedicated to manage the already

described I/O image operations and to perform other functions like :
-Formatting image data in order to fit different storage organization like mass or Host memory respect to that of the bit-plane.
-Processing of images with dimensions greater than the pyramid section size.
-Acquisition and storing of images coming from the external world.
-Trasfering processed images from pyramid to the external world.

3. CONTROLLER UNIT PHILOSOPHY

A number of implementation modes can be thought, each having advantages and disadvantages. In the following we will present some example.

3.1 HOST COMPUTER USED AS CONTROL UNIT

In this mode no specialised hardware is required besides that supplied with standard general purpose computers (HOST). The Image Analysis (IA) algorithms, compiled in the Host, consist of a sequence of elementary instructions that are sent to the multiprocessor parallel unit, interleaved with a sequence of instructions executable by the Host itself as data image input/output operations.

ADVANTAGES are obvious:
-Host Operating System is well known.
-High level languages are available.
-No specialised hardware is required.
-Easy and fast implementation is possible.

DISADVANTAGES are :
To maximize throughput (number of elementary intructions to the multiprocessors unit and data I/O), the total Host power must be dedicated only to the current IA task. Note that, also in this case, the throughput of whatever general computer, will not be enough. In fact the time between two consecutives elementary instructions for PAPIA is one microsecond. This word-rate is reached, by a general purpose computer, only working in Dyrect Memory Acces (DMA) mode. But DMA cannot be utilized for the entire program, if real time flow control is necessary; at that time the flow must be stopped and, after some computations, on processed data, decision is taken and the flow restarts. The waste in time, as can be easily understood, is so that the advantage of using a parallel architecture is lost.
Another disadvantage is that the emulation of several control unit in a MSIMD environment is wasteful in time; it can end in a useless effort.
Nevertheless such an implementation can be used when speed is not the goal; performance evaluation of a new parallel machine in almost real environment (emulation) and

implementation of IA primitives are examples of its use.

3.2 HOST COMPUTER + EXTERNAL HARDWARE USED AS CONTROL UNIT

The goal of this mode of implementation is to overcome the already discussed disadvantages i.e. lost of speed.

The idea is to use the Host computer only for tasks that are not related with the flow control of elementary instructions, but for data aquisition, high level software implementation, data and program storage, image presentations.
Between Host and the multiprocessor parallel unit, two other units are interposed; one sequential and one elementary intruction feeding unit. The last two units work in almost asyncronouse mode respect each other and the host activities i.e. in a pipe mode. Different languages in each unit are used: the Host manages a high level language, the sequential unit a medium level and the last unit microprogrammed language. The IA problems are so executed in a such a way that the time for each level processing is nearly the same. In this mode bottlenecks are removed. Remembering that IA can be distinct in two parts : one sequential and one parallel, it is obvious that in this mode each part has the natural processing unit.

ADVANTAGES of this mode are already explaned. Here we stress that the Host power demands are less pressing.

DISADVANTAGES are related with realization difficulties; they are:
-Different specialized hardware units must be implemented.
-Different no standard languages must be used.
-Piped processing units must be syncronized.
-Cost and time effort is high.

4. HARDWARE FOR THE PAPIA CONTROL UNIT

The last approach has been followed in designing the PAPIA control unit. So two units are used; namely: The sequential, from now on quoted as Global Control and Scalar Processing Unit (GCSPU) and the second quoted as Microprogrammed Control Unit (MCU)

GLOBAL CONTROL AND SCALAR PROCESSING UNIT

The GCSPU has been developped using a Philips PG 2000 model Single Board Computer (SBC), based on the Motorola MC 68000 microprocessor, interfaced to a VME standard bus. Memory space for ROM, RAM, two serial and one parallel port is provided on board. On of-board VME memory space have been mapped: external RAM, two parallel port controllers and a number of input registers housed in the MCU.

A 512 words 16-bits wide FIFO memory has been mapped on the board memory.

A monitor firmware in the SBC allows operators to control the unit activities interacting by a video terminal and keyboard.

The Host interface is implemented by interrupt driven parallel 16-bits input/output port.

The FIFO queue, and the several memory mapped registers interface the GCSPU to MCU. The asyncronous activites of the two controller unit are syncronised by vectorized interrupt requests coming from MCU and by the FIFO empty flag line.

The firmware consists of a parametric macroroutines collection and other utilities. The Host program will be a series of instructions for the SBC microprocessor, a lot of them being parametric macroroutine calls. Macroroutines can call each other. The aim of a macroroutine is to execute, on the SBC, the sequential and scalar part of the IA algorithm and to call microprogrammed parametric microroutines (IA primitives) residing on the MCU, by FIFO queue.

Fig. 1 is a block diagram of the overal system controller with the GCSPU elements evidenced.

In conclusion, we stress that only one GCSPU unit will be enough to drive several MCUs when the pyramid will be used in MSIMD mode.

MICROPROGRAMMED CONTROL UNIT

The difference between implementation modes of a central control unit, is essentially due to this unit. In fact emulation of the GCSPU by the host is easily implemented with no time execution penality, only more partecipation is required to the Host.

The MCU has been developped around the Advanced Micro Devices (AMD) Sequencer and other related electronic components, bit slice series, in order to realize a microprogrammed unit.

A random logic hardware approach realizes a machine faster than that microprogrammed but changes are very difficult to implement. On the other side microprogramming provides a more modular and systematic method of implement the control mechanism, because the control sequence is stored in an array that can be accessed systematically. Changes are easy to implement being software changes. Customizing the unit to suit various applications becomes easy, fast and efficient.

The heart of a microprogrammed machine, the control path, comprises:

-The sequencer, the microprogram memory and the pipeline register.

In the following a coarse description will be given with some implementation and use notes.

4.1 THE PIPELINE REGISTER

It is essentially a fast register, clock strobed, connected at the output of the memory. It is used as storage of control word for the sequencer. This is executed in the current cycle, while the next microaddress is simultaneusly computed in the sequencer, which then accesses the contents of this address in the control memory.

4.2 THE MICROPROGRAM MEMORY

It stores the control bits for each microinstruction. It is usualy a PROM or an electrically programmable read only memory (EPROM) or a Random access memory (RAM). If a RAM is used, it is referred to as Writable Control Store (WCS). In our first implementation a WCS, 4Kw, has been used because it is a development system and because it is useful, for program debugging and for context switching, changing the microcode.

The WCS is loaded at power up and reset time with standard or user supplied software. Standard software is a microprogrammed IA primitives collection and various utilities. A memory address corresponds to 80-bits word groupped in various fields, each controlling a specialized part of the MCU.

4.3 THE SEQUENCER

The AM 2910A microprogram controller has been used.

It essentially controls the sequence of execution of microinstructions stored in the microprogram memory. Besides the capability of sequential access, it provides conditional branching to any microintructions within the 4096-microword range. A last-in first-out stack provides microsubroutines return linkage and looping capability; there are nine levels of nesting of microsubroutines. Microintruction loop count control and loop iterations are provided with a count capacity of 4096 by an internal pre-settable counter/register.
Sixteen powerful microinstructions, most of which are conditional on external condition input (CC), state of internal loop counter or both, are implemented. 100-ns cycle times are supported.

5. SPECIALIZED MCU HARDWARE IMPLEMENTATION

Around the presented control mechanism we have made our unit adding specialized hardware. The block diagram is given in fig. 2

The guideline for understanding it is the following:

Requests coming from the FIFO, are implemented by a sequence of a number of 16-bit words. The first word is the routine entry point (Instruction word), the following one are parameter values used and loaded by the called routine itself, in the provided registers.

The Instruction word is stored in the intruction register (IR), parameters are stored each in one of the following registers:

-Three address registers
-One 16-bits (external counter buffer register).
-One 12-bits (internal counter/register buffer).
-Two interconnected 16-bits Parallel-in Serial-out shift registers (PISO)

Each listed register inputs data from the "internal data bus". Only the FIFO buffer can put data on it. At the time that a load register instruction is executed FIFO control of

data ready line is performed by MCU. In case FIFO data is not ready the MCU stops its activities waiting for data. Syncronization between GCSPU and MCU activities is so obtained.

One Up/Down presettable (address) counter, with three state output, is at the output of each buffer register. Updated contents can be used for counting operations, loop iterations and variable addresses to the pyramid PEs private memory or near neighborhood internal registers.

Counter flags for zero or positive counter values detection plus outcoming bit from each PISO sections are multiplexed and feed to the condition code sequencer input, in order to implement the microintructions control flow.

The PISO serial output is feeded also to the pyramid apex when a pyramid imput operation is performed. In this way a costant can be broadcasted to all enabled PEs of the pyramid.

A Serial-in Parallel-out 32-bits shift register (SIPO) (two sections) with three state output is used to receive the pyramid apex outcoming bits. In addition it is simultaneusly sent to the Condition Code Multiplexer for flow control.

At the output of each address counter, a buffer register with three state output is connected. They are loaded, with the related counter content, at each output counter operation.

So, syncronization between MCU and GCSPU is performed using vectorized interrupt requests.

The three state output of the buffer registers now described, PISO, SIPO and Interrrupt Vector Register (IVR) (see later) are connected at the data bus of the GCSPU microprocessor, so they can be read by it when an interrupt request comes from MCU.

All described special components are controlled in parallel by the control word at the output of the pipeline register.

6. MICROASSEMBLER INSTRUCTIONS

The microassembler intruction is formatted by one line divided in three different fields.

The first is the label field. The mnemonic label must be terminated with the special character ":".

The second is the instruction field. In this, different instructions can be interlinked putting the special character & between. Each subinstruction is composed by the operation mnemonic followed by a number of related arguments. There are four subinstrution categories each related with a special part of the MCU.

1) Register and pyramid instructions.

In the case a pyramid intruction opcode [7] has a variable part, like memory o n.n. addressing, it can be choosed to be the output of an address counter. The number of bits requested in overlay are indicated in the same intruction. The advantage of using such an implementation is evident; parametric microroutines can be used.

2) Sequencer instructions

3) Counters instructions

4) Counters flag output instructions

In general, when an intruction is sent to the pyramid the MCU performs a status pyramid clock control. If the pyramid

clock is not ready, the MCU stops its activities, waiting for a ready status. So the syncronization between MCU and pyramid activities is realised.

The third field is the comment one. It must be started with the special character ";"

7. SOFTWARE TOOL FOR THE MICROPROGRAMMING MACHINE

The assembler text program is processed by an general purpose meta assembler, commercially available from Microtec Research Company, (META29R). It has three fundamental sections:

1)Definition section; It allows intructions and related bit-map definition, in order to customize the assembler instructions.

2)Relocatable meta assembler section; Its output is the assembled program as binary codes with relocatable addresses and cross references tables.

3)Linker loader section; Several assembled programs can be linked together and an absolute coded binary program is produced ready to be loaded and executed by the MCU.

Acknowledgements:

The arguments here reported and related to the controller realization, come from the contribution of Palermo-IFCAI/CNR electronic laboratories tecnical staff. In particular the author is deeply indebited to Mr. O. Catalano for his collaboration in design and testing the overal system and to Mr. F. Nicotra for the carefull wiring of microprogrammed section. General idea comes from deep discussions with the PAPIA collaboration group quoted in [5].

This work has been made sub CNR contract (IFCAI-Palermo Institut) and partially supported by Comitato Regionale Ricerca Nucleare e Struttura della Materia of Government of Sicily.

References

[1] Selim G. Akl, "Parallel Sorting Algorithms", Academic Press, 1983

[2] V. Cantoni, M. Ferretti, S. Levialdi, F. Maloberti, "A pyramid project using integrated technologies" , Edit. S. Levialdi Academic Press, New York pp. 121-132, 1985

[3] O. Catalano, G. Gerardi, A. Machi', "The PAPIA hardware control", III int. conf. on image analysis and processing, Rapallo, 1985 (Proceedings to appear)

[4] V. Cantoni, L. Carrioli, L. Cinque, M. Ferretti, S. Levialdi, A. Machi', "Parallel image processing primitives", III int. conf. on image analysis and processing, Rapallo, 1985 (Proceedings to appear)

[5] V. Cantoni, L. Carrioli, O. Catalano, L. Cinque, V. Di Gesu', M. Ferretti, G. Gerardi, S. Levialdi, R. Lombardi, A. Machi', R. Stefanelli, "The PAPIA image Analysis System", 2nd International Technical Symposium on Optical and Electro-Optical Applied Science and Engineering, Cannes, December 1985.

[6] G. Gerardi, O. Catalano, "Descrizione funzionale dell'unita' di controllo di PAPIA -Manuale di microprogrammazione", IFCA/CNR Internal report n. 05, Apr. 86.

[7] J. Mick and J. Brick, "Bit slice microprocessor design", Mc Graw Hill, 1980.

[8] " VMEbus CPU Board PG2000 user manual"
Phylips, 1984.

[9] "Bipolar Microprocessor Logic and Interface Data Book"
Advanced Micro Devices, 1985.

[10] "Relocatable Macro Meta Assembler and Linking Loader"
Microtec Research, 1984.

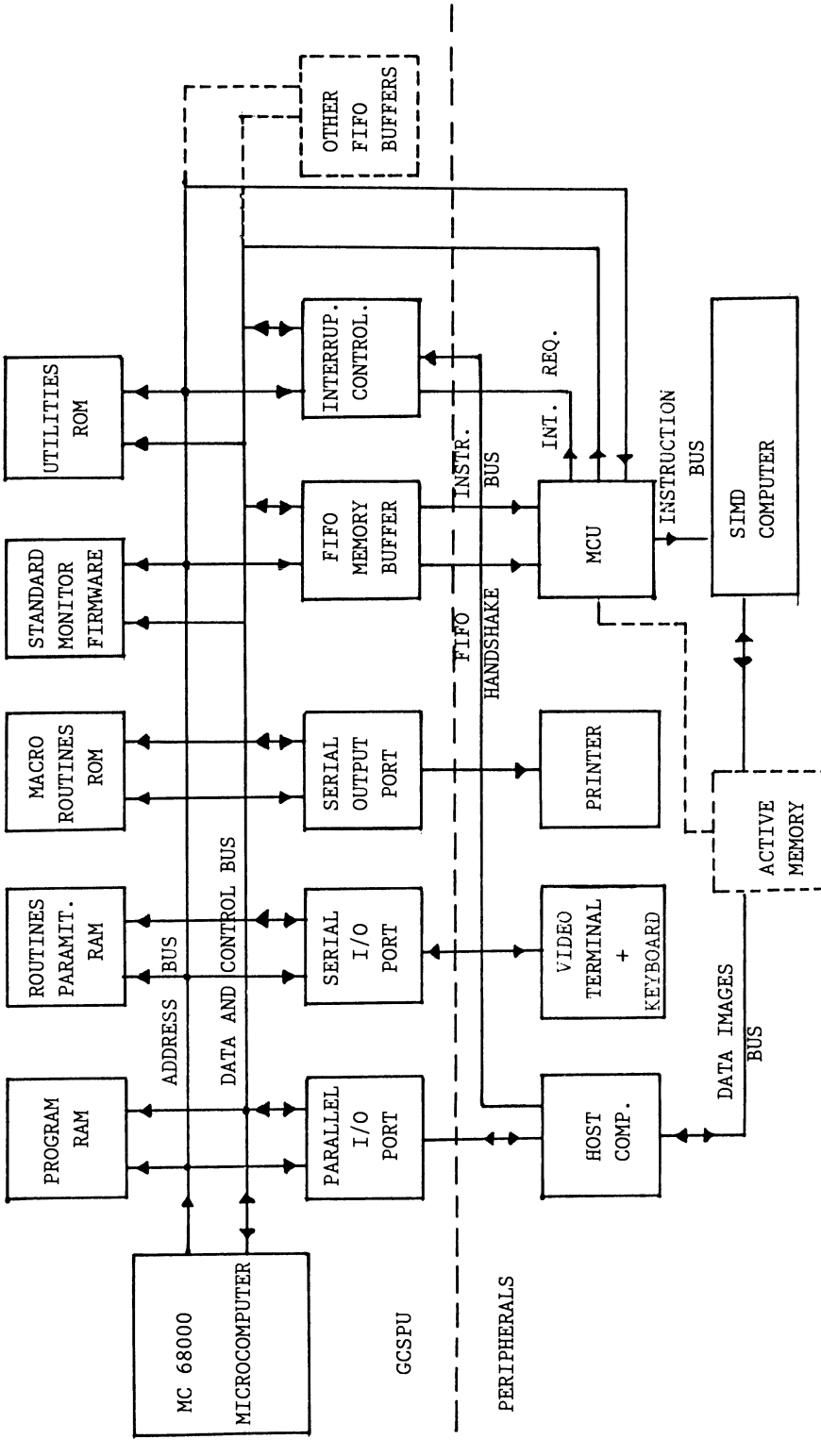


Fig. 1 GLOBAL CONTROL AND SCALAR PROCESSING UNIT

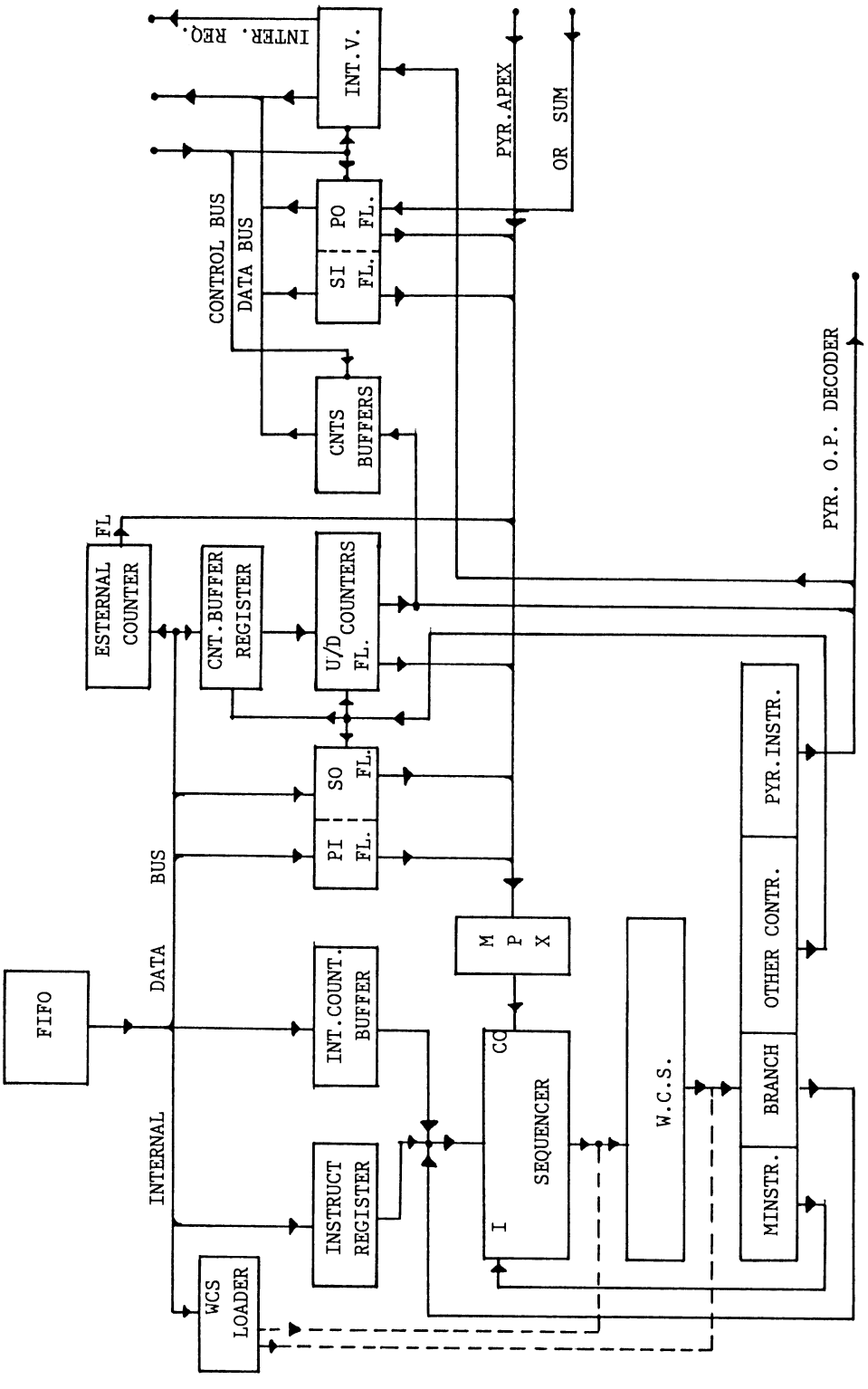


Fig. 2 MICROPROGRAMMED CONTROL UNIT BLOCK DIAGRAM

CUSTOM MADE PYRAMIDS

Shmuel Peleg
Orna Federbusch

Dept. of Computer Science
The Hebrew University of Jerusalem
91904 Jerusalem, Israel

ABSTRACT

Pyramids are data structures used to store and process images in multiple resolutions. The bottom level of the pyramid usually stores the original image, while higher levels of the pyramid represent the image in reduced resolutions. The reduction of image size between levels is usually by a factor of two, and identical factors are used to reduce each level to the higher level.

This paper describes a scheme that enables the pyramid to be adapted to different applications. The pyramid can be reduced arbitrarily between levels from any given rectangular size to any desired rectangular size. Also, the reduction factor can be made adaptive to region properties, enabling smooth regions to be reduced more than "busy" regions.

1. Introduction

In the most common pyramid scheme each pixel in a given level represents a block of pixels from the level below. A survey on pyramids can be found in [1].

In this paper a method is described to construct pyramids of arbitrary size at each level, so that resolution can be reduced as needed and not only by fixed powers of two. Further, the reduction from level to level does not have to be uniform over the entire image, and can vary based on local image properties.

The basic step in the proposed pyramid construction is a spatial resampling technique used in graphics [2,3]. This transformation is carried out in two passes, where the first pass affects only rows, and the second only columns. As each pass processes a one dimensional entity, the method will be described in one dimension.

The spatial resampling has the desirable property that all input pixels fully contribute to the output pixels with no gaps or overlaps, therefore minimizing sampling artifacts.

1.1. One Dimensional Contraction-Expansion

Given is a vector $V = (v_1, \ldots, v_N)$ of N pixels, to be changed to a vector $W = (w_1, \ldots, w_K)$ of length K. Each pixel in V contributes $f = K/N$ to pixels in W, such that the total contribution to each $w_i \in W$ is one. Also, if v_i contributes to w_j, and $k > i, l < j$, then v_k does not contribute to w_l (no crossovers). For example, to reduce (v_1, v_2, v_3, v_4) to (w_1, w_2, w_3), $f = 3/4$, and the reduction is as shown in Figure 1.

This method has the following features:

This paper has been supported by a grant from the Israel Academy of Sciences.
Author's electronic address: peleg@hujics.bitnet

NATO ASI Series, Vol. F 25
Pyramidal Systems for Computer Vision
Edited by V. Cantoni and S. Levialdi
© Springer-Verlag Berlin Heidelberg 1986

$$w_1 = \frac{3}{4}v_1 + \frac{1}{4}v_2$$

$$w_2 = \frac{1}{2}v_2 + \frac{1}{2}v_3$$

$$w_3 = \frac{1}{4}v_3 + \frac{3}{4}v_4$$

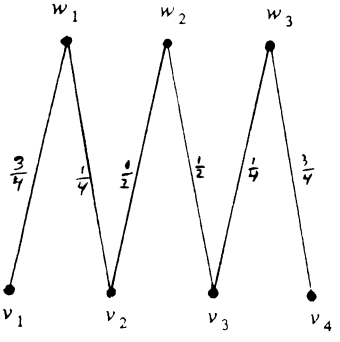


Figure 1: Reduction from a level of size 4 to a level of size 3.

- Each pixel of the input vector has equal contribution to the output vector.
- There are no border effects, as there is no need to use values of pixels outside the image boundary.

1.2. One Dimensional Weighted Resampling

In the resampling of the previous section all input pixels had the same contribution f to the output pixels. The algorithm can be adapted to the case where contribution can vary, and each pixel v_i is assigned a "forward weight" $f(v_i) > 0$, such that $\sum_{i=1}^{N} f(v_i) = K$, where K is the number of the pixels in the output vector W.

We construct W using the principle of the previous section, except that each pixel v_i has his own weight $f(v_i)$. For example, when $V = (v_1, v_2, v_3, v_4)$, with weights $f(v_1) = 1$, $f(v_2) = 1$, $f(v_3) = 0.5$, $f(v_4) = 0.5$, is to be reduced to $W = (w_1, w_2, w_3)$, the reduction is as shown in Figure 2.

$$w_1 = v_1$$

$$w_2 = v_2$$

$$w_3 = \frac{1}{2}v_3 + \frac{1}{2}v_4$$

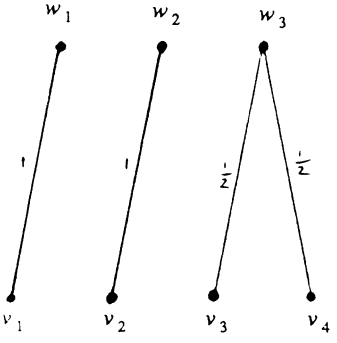


Figure 2: Weighted reduction, with weights as marked, from a level of size 4 to a level of size 3.

2. Pyramid Construction

By use of the contraction method in Section 1.1, an input image of any size $M \times N$ can be reduced to an output image of any desired size $K \times L$. Through repetition of this process (each level being of any desired size) a complete pyramid can be built. In Figure 3 a one-dimensional example of the construction of a three level pyramid is presented. Note that the contribution of bottom level to top level varies with the properties of intermediate levels. Higher levels are in general weighted averages of bottom level pixels, where the overlap between regions included in the averages increases with the number of intermediate levels.

This pyramid scheme has the following properties:

- Each pixel has equal contribution to the level above.
- Reduction can be made from a level of any given rectangular size to any desired rectangular size.
- Pixels in higher levels of the pyramid are weighted averages of pixels in the bottom level, with the possibility of overlap: a pixel in the bottom level can contribute to several pixels in any given higher level. This property is similar to that of Burt's overlapping pyramids [4].
- The degree of overlap depends mostly on the number of intermediate layers.

An example of an image embedded in such a pyramid is shown in Figure 4.

These pyramids can be used for same applications as Burt's overlapping pyramids [4]. We also follow Burt by construction of "Laplacian Pyramid", where each level L_i in the Laplacian Pyramid is the difference between levels G_{i+1} and G_i of the original pyramid. Level G_{i+1} is enlarged thru the same resampling technique to the size of level G_i before performing the subtraction. As in Burt's Laplacian Pyramid, each level in L_i is a band-pass filter as a difference of two low pass filters.

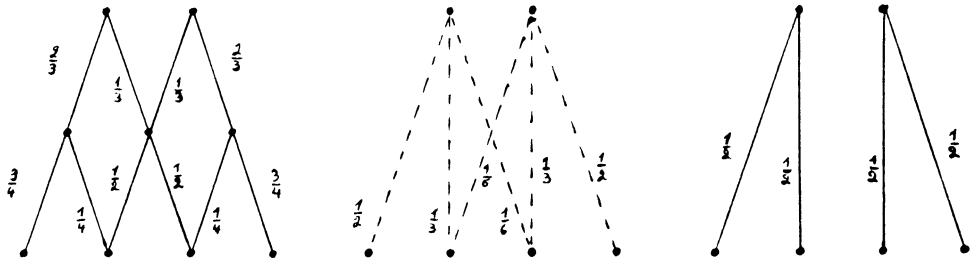


Figure 3: One dimensional pyramid.

a) Three levels, 1-D pyramid of size 4-3-2. A number on an arc is the contribution of the lower pixel to the upper pixel.

b) The overall contribution of the lower level to the top level for the pyramid (a).

c) A pyramid that reduces from base of size 4 to size 2 directly.

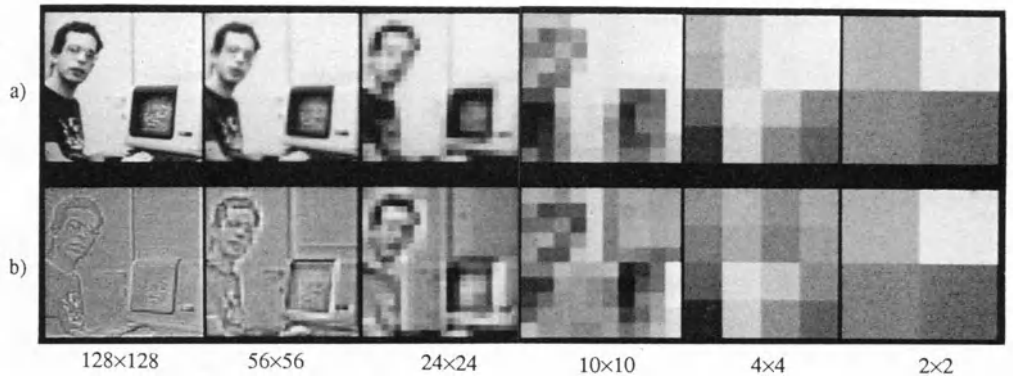


Figure 4:

An image embedded in a 6-level pyramid with level sizes as marked. Levels were enlarged to original size for visibility.

a) Levels of the pyramid. The leftmost image is just the original; successive pictures are the reduced images at higher levels.

b) Levels of the Laplacian Pyramid (see text), where each level is a band-pass filter. Each image is the difference between the pyramid image of same size, and the image at the level above (expanded). Zero is represented as gray, negative parts are lighter and positive parts are darker.

3. Adaptive Resampling

The spatial resampling scheme employed in the pyramid construction allows us to vary the resampling at each region of the image, rather than the uniform resampling common in existing pyramid schemes. Non-uniform sampling has the advantage that regions with "interesting" activity can be resampled more densely than "uninteresting" regions. This effect can also be observed in human perception where a segment with higher "busyness" seems longer than a straight line segment [5], as in Figure 5. This might indicate that more storage is used in the brain for the divided segment.

To achieve the effect of adaptive sampling, we need to use a "busyness" measure. Measures could be selected to best fit an application, and in this paper we use the smoothed absolute value of the Laplacian.

3.1. One Dimensional Adaptive Pyramid

In the one dimensional case, we first apply a Laplacian to the vector V of length N, take its absolute value, and smooth the result. The weights are then normalized to get the "forward weights" f_i $(1 \le i \le N)$ such that $\sum_{i=1}^{N} f_i = K$, where K is the number of pixels in the output vector W. The weighted resampling of Section 1.2 is then applied. While resampling, each pixel at the output vector W obtains a "backward weight" b_i $(1 \le i \le K)$, which indicates how many pixels of the lower levels contribute to it. $\sum_{i=1}^{K} b_i = N$,



Figure 5:
An illusory distortion in which the divided space appears longer than the undivided space (Oppel-Kundt illusion).

and the b_i weights will be used as weights when reconstructing a level from the higher levels.

Example 1: In Section 1.1 (Figure 1) we used equal contributions of first level pixels to the second level:

$$f(v_1) = f(v_2) = f(v_3) = f(v_4) = \frac{3}{4}.$$

This resulted in the following construction for the second level: $w_1 = \frac{3}{4}v_1 + \frac{1}{4}v_2$, $w_2 = \frac{1}{2}v_2 + \frac{1}{2}v_3$, $w_3 = \frac{1}{4}v_3 + \frac{3}{4}v_4$. As w_1 was built from all of v_1 and $\frac{1}{3}$ of v_2, its "backward weight" $b(w_1) = 1\frac{1}{3}$; w_2 was built from $\frac{2}{3}$ of v_2 and $\frac{2}{3}$ of v_3, yielding $b(w_2) = 1\frac{1}{3}$; similarly, $b(w_3) = 1\frac{1}{3}$. Therefore, when building V back from W, equal weights will be used.

Example 2: In Section 1.2 (Figure 2) we had a weighted construction with the first level weights:

$$f(v_1) = 1\ ,\ f(v_2) = 1\ ,\ f(v_3) = 0.5\ ,\ f(v_4) = 0.5.$$

The resulting construction has been $w_1 = v_1$, $w_2 = v_2$, $w_3 = 0.5\,v_3 + 0.5\,v_4$. In this case we have $b(w_1) = b(w_2) = 1$ as each is being supported by one element in V, while $b(w_3) = 2$ as w_3 is being supported by both v_3 and v_4. Using the backward weights, when the V level will be expanded from the W level we will use $v_1 \leftarrow w_1$, $v_2 \leftarrow w_2$, $v_3 = v_4 \leftarrow w_3$.

3.2. Two Dimensional Adaptive Pyramids

We build a two dimensional adaptive pyramid using the same principle as the one dimensional case of the previous section. However, if each row or column will be processed independently, the image will loose its structure in the sense that connectivity will not be preserved. This can be caused when the weights of each row or column are computed independently of the neighboring rows/columns. As the loss of connectivity structure is unacceptable, we select a scheme where all rows and columns use the same weights. In this scheme, row weights are computed based on average busyness for all rows, and column weights are computed based on average busyness for all columns. The following examples will show the

reduction from 4×4 to 2×2, given the computed busyness measures.

Example 3: Given the following busyness matrix

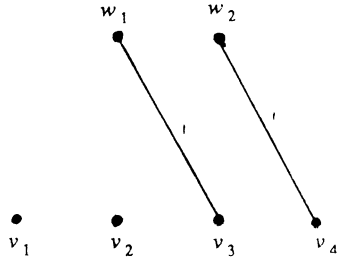
$$\begin{bmatrix} 0 & 0 & 2 & 2 \\ 0 & 0 & 2 & 2 \\ 2 & 2 & 0 & 0 \\ 2 & 2 & 0 & 0 \end{bmatrix}$$

The average row busyness is (2 , 2 , 2 , 2), yielding row weights of (0.5 , 0.5 , 0.5 , 0.5). Same is true for the column weights. This will result in a reduction of the 4×4 image to a 2×2 image, where each output pixel is the average of a 2×2 block in the input image.

Example 4: Given the busyness measure

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 2 \\ 0 & 0 & 2 & 2 \end{bmatrix}$$

the average row busyness is (0 , 0 , 2 , 2) yielding row weights of (0 , 0 , 1 , 1), and the same happens for the columns. The row and column reduction will both be:



and given the image

$$\begin{bmatrix} v_{11} & \cdots & v_{14} \\ \cdot & & \cdot \\ \cdot & & \cdot \\ \cdot & & \cdot \\ v_{41} & \cdots & v_{44} \end{bmatrix}$$

the reduction will be:

$$\begin{bmatrix} v_{11} & \cdots & v_{14} \\ \cdot & & \cdot \\ \cdot & & \cdot \\ \cdot & & \cdot \\ v_{44} & \cdots & v_{44} \end{bmatrix} \xrightarrow{\text{row reduction}} \begin{bmatrix} v_{13} & v_{14} \\ \cdot & \cdot \\ \cdot & \cdot \\ \cdot & \cdot \\ v_{43} & v_{44} \end{bmatrix} \xrightarrow{\text{column reduction}} \begin{bmatrix} v_{33} & v_{34} \\ v_{43} & v_{44} \end{bmatrix}$$

The effects of embedding an image in an adaptive pyramid can be seen in Figure 6.

4. Concluding Remarks

A pyramid scheme which is adaptive to the reduction of resolution both between levels and also spatially within levels has been presented. Such pyramids can be of use when many levels of resolution are

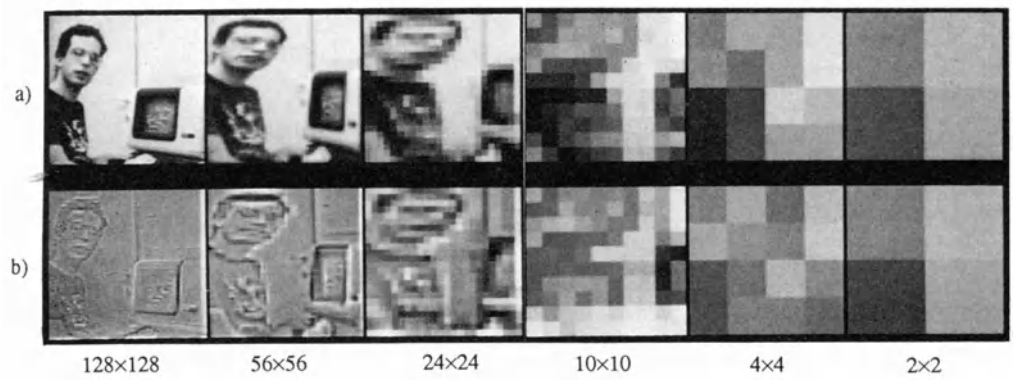


Figure 6:
6-level adaptive pyramid, with level sizes as marked. Levels were enlarged to original size for visibility.

a) Levels of the adaptive pyramid.

b) Levels of the Laplacian Pyramid.

needed, and may help in compression when using denser resampling for busy regions.

5. References

1. A. Rosenfeld (Ed.), *Multiresolution Image Processing and Applications,* Springer, 1984.

2. K. Fant, A nonaliasing, real-time spatial transform technique, *IEEE Computer Graphics and Applications,* Vol 16, January 1986, 71-80

3. E. Catmul and A. R. Smith, 3D transformations of images in scanline order, *Computer Graphics (Proc. SIGGRAPH 80),* Vol 14, July 1980, 279-289.

4. P. J. Burt and E. H. Adelson, The Laplacian pyramid as a compact image code, *IEEE Trans. Communications,* Vol COM-31, 1983, 532-540.

5. S. Coren and J. S. Girgus, *Seeing is Deceiving,* Lawrence Erlbaum Associates, 1978.

PARADIGMS FOR PYRAMID MACHINE ALGORITHMS[1]

Steven L. Tanimoto

Dept. of Computer Science, FR-35
University of Washington
Seattle, WA 98195/USA

Abstract

While the development of algorithms for parallel computers is often difficult, in the domain of image processing one may often use an appropriate methodology of algorithm design that greatly simplifies such development. The highly parallel "pyramid-machine" architecture supports several styles of programming for parallel machines, and it allows complicated transformations of images to be computed very rapidly. After alternative classification schemes are given, several powerful paradigms for pyramid-machine algorithms are described, including pyramid-building, tree-search, propagation, relaxation, and parallel subpyramids. The impacts of design methodology and the software development environment on the algorithm-design process are also discussed.

[1]Research supported in part by NSF Grant MCS-8310410

NATO ASI Series, Vol. F 25
Pyramidal Systems for Computer Vision
Edited by V. Cantoni and S. Levialdi
© Springer-Verlag Berlin Heidelberg 1986

1 Introduction

1.1 Image-Processing Problems

The need to rapidly analyze images arises in several applications areas including industrial parts inspection, visual navigation systems for robots, and medical diagnosis from real-time imaging devices. Such applications place heavy computational demands on computers. In addition, the solutions to such problems are often difficult to formulate, even disregarding the speed considerations.

1.2 Need for Parallel Algorithms

The great computational demands strongly suggest that parallel processing be incorporated in systems that perform high-speed image analysis. Not only are many processors or processing elements needed, but there is a need to interconnect them in a fashion that permits them to communicate gracefully during the course of a computation.

1.3 Difficulty of Designing Parallel Algorithms

It has proved difficult to design effective parallel algorithms for many problems in computation. For example, the "unification" process in logic programming seems to be inherently rather sequential. Even when there clearly exists the potential for applying parallelism to a problem, formulating an algorithm which embodies the parallelism may be rather difficult. If one uses a parallel computer that permits many independent processes to progress simultaneously, the programmer or algorithm designer must spend much time coordinating the processes, defining synchronization protocols, for example.

1.4 Need for Good Tools and Methodology

With appropriate tools, many of the difficult steps in designing parallel algorithms can be avoided or simplified. For example, standard patterns of synchronization may be studied in advance and incorporated into algorithms. By adopting certain methodologies for the design of algorithms, it may be possible to simply provide a "generator" program with some specific parameters, and it will produce a program meeting the needs of the application.

2 Pyramid-Machine Architecture

2.1 Brief Motivation

The pyramid-machine architecture may be regarded as a combination of the architectures of cellular arrays and tree machines. It is commonly suggested

that in order to get ample computational power for processing a digital image, one may assign a processor to each cell (pixel) of the image, and allow the processors to communicate directly with their immediate neighbors in the array [Duff 1976]. This model is implicit in most cellular-logic approaches to image processing [Preston et al 1979]. While such a distribution of processors is an important improvement over one processor or a row of processors, it still does not provide much support for the computation of global (non-local) characteristics of an image. In order to ameliorate the effects of this limitation, hardware based on the processor-per-pixel approach has included special attachments to compute sums over all cells and the inclusive OR of a signal over all the cells, thus giving some global capabilities to the system [Duff 1976].

A hierarchical structure can retain most of the advantages of the processor-per-cell arrangement, while obtaining a capability for global computations that is mathematically elegant. Pyramidal structures that are simultaneously parallel and serial allow gradual formation of more and more global descriptions of image data, in parallel [Uhr 84].

Some of the advantages of pyramid machines for image processing derive from the multi-resolution data structure that is implicit in these systems [Rosenfeld 1984], [Tanimoto and Pavlidis 1975]. Others result from the short (log N) data paths from the pixels to the roots of the pyramids [Tanimoto 1981]. Also, operations such as median filtering can be efficiently performed [Tanimoto 1983c]. Some pyramid machines currently under development are described in [Cantoni et al 1985], [Schaefer et al 1985], and [Tanimoto et al 1986].

2.2 Cells, Hierarchical Domains, and Pyramids

The principal data structures manipulated by pyramid machines are "pyramids." A pyramid may be regarded as a collection of images of a single scene at different resolutions. However, we present a more formal definition that allows a more precise discussion of pyramids and the machines and algorithms which manipulate them.

We define a cell as a triple of integers. The general cell, (k, i, j), occurs at "level" k, "row" i, and "column" j.

For a positive integer L, the *hierarchical domain* with $L+1$ levels is the set of cells

$$\{(k,i,j) \text{ such that } 0 \leq k \leq L, \text{ and } 0 \leq i \leq 2^k, \text{ and } 0 \leq j \leq 2^k\}.$$

The *kth level* consists of those cells of the hierarchical domain whose first coordinate is k. The number of cells in level k is 4^k. The largest level, level L is called the *base*, or the *finest level.* Level 0 consists of the single cell $(0,0,0)$ and is called the *root.*

A *pyramid* is any function whose domain is a hierarchical domain. If the range of the function is $\{0,1\}$ then the function is called a *binary pyramid* or

a *bit pyramid.* If the range is $\{0,\ldots,255\}$ then it is a *byte pyramid.* Pyramids with somewhat different but related definitions can be found in the literature [Tanimoto and Pavlidis 1975], [Burt 1979].

Each cell of a hierarchical domain has a neighborhood of cells considered to be adjacent to it. This neighborhood generally consists of cells on three levels: the level containing the cell, the level above (level $k-1$), and the level below ($k+1$). Cells that are in the base, at the sides or at the root have incomplete neighborhoods. In order to simplify the discussion, we pretend that "dummy cells" exist around the outside of the hierarchical domain thereby completing the neighborhoods of border cells.

The neighborhood for each cell contains thirteen cells (some possibly dummies). These are are defined for the general cell (k,i,j) as follows:

N_1	"parent"	$(k-1,\lfloor i/2\rfloor,\lfloor j/2\rfloor)$
N_2	"northwest"	$(k,i-1,j-1)$
N_3	"north"	$(k,i-1,j)$
N_4	"northeast"	$(k,i-1,j+1)$
N_5	"west"	$(k,i,j-1)$
N_7	"east"	$(k,i,j+1)$
N_8	"southwest"	$(k,i+1,j-1)$
N_9	"south"	$(k,i+1,j)$
N_{10}	"southeast"	$(k,i+1,j+1)$
N_{11}	"northwest son"	$(k+1,2i,2j)$
N_{12}	"northeast son"	$(k+1,2i,2j+1)$
N_{13}	"southwest son"	$(k+1,2i+1,2j)$
N_{14}	"southeast son"	$(k+1,2i+1,2j+1)$

Together with the home cell, that is, (k,i,j), which may be labelled N_6, the pyramid neighborhood comprises fourteen cells. The nine of these cells in level k make up the *lateral neighborhood.* The remaining five cells are the *quadtree neighbors* or the *hierarchical neighbors.*

Each cell has three siblings. The siblings of a cell are the other three sons of the cells parent. The coordinates of the siblings of a cell (k,i,j) can be obtained by complementing the least significant bit of the row coordinate (i) and of the column coordinate (j); the three combinations different from the original give the three siblings.

It is interesting to note that hierarchical domains may be defined analogously for base levels that are not square grids, or in such a fashion that the levels taper more gradually than is the case here [Tanimoto et al 1984].

2.3 Operations

It is generally assumed that a pyramid machine contains many processing elements ("PE's"), one per cell of a hierarchical domain. The capabilities of these

processing elements differ with the particular pyramid-machine proposal. We shall describe one set of operations that is reasonably powerful and yet relatively economical to implement.

The most interesting operation in our set is a general matching operation. We describe its behavior in terms of the bit pyramids it takes as input and produces as output.

Let X be a bit pyramid. Then, given any cell (k,i,j), the vector of X values for the neighborhood of the cell is

$$\nu_X = [X(N_1), X(N_2), \ldots, X(N_{14})].$$

In the case that N_i happens to be a dummy cell (because of the case when the cell in question lies on the border), then we define $X(N_i)$ to be 0.

Let P be a "pattern" consisting of fourteen symbols, each of which is either 0, 1, or D. That is,

$$P = [P_1, P_2, \ldots, P_{14}]$$

where P_i is 0, 1, or D. Although a pattern is a vector of fourteen elements, we shall write it as a string of fourteen symbols, to save space. The AND_Match of P with ν_X is given by

$$\bigwedge_{n=1}^{14} (P_i \wedge^* X(N_i))$$

where

$$x \wedge^* y \text{ is } \begin{cases} 1, & \text{if } x = D \text{ or } x = y; \\ 0, & \text{otherwise.} \end{cases}$$

The OR_Match of P with ν_X is given by

$$\bigvee_{n=1}^{14} (P_i \vee^* X(N_i))$$

where

$$x \vee^* y \text{ is } \begin{cases} 1, & \text{if } x = D \text{ and } x \neq D; \\ 0, & \text{otherwise.} \end{cases}$$

Each of these operators is extended to operate on the entire pyramid X by applying it to the neighborhood of each cell. Thus we may write

$$Y = \text{AND_Match}(P, X)$$

and

$$Z = \text{OR_Match}(P, X)$$

where Y and Z are the pyramids whose values are the results of matching each neighborhood vector of X with P, using AND_Match and OR_Match, respectively.

By suitably choosing the values of P, the matching operator can be made to perform several useful functions directly. Some examples are shifting a pyramid in one of the eight lateral directions, and detecting local extremities and local concavities in a shape.

If we take $P = DDDDDD1DDDDDDD$, and assign a name to the resulting function of X, we obtain:

$$\text{ShiftWest}(X) = \text{AND_Match}(DDDDDD1DDDDDDD, X).$$

This function happens to be equivalent to

$$\text{OR_Match}(DDDDDD1DDDDDDD, X).$$

since in either case, the effect is to copy into a cell of Y, the value of its neighbor to the east in X. Similar shifting functions can be defined for the other seven lateral directions.

A local extremity of a pattern of 1's in a 2-dimensional array may be defined as a 1 surrounded by 5, 6, or 7 contiguous 0's. For example, an extremity surrounded five 0's is shown below:

1	0	0
1	1	0
1	0	0

Thus defined, there are 24 different configurations for extremities, taking the eight possible orientations for each of the three kinds into account. A function which detects the case illustrated is $\text{AND_Match}(D100110100DDDD, X)$.

Local concavities may also be defined on the basis of 3 by 3 arrays. One kind of concave configuration is described by:

D	1	D
D	0	D
D	1	D

The detection of this set of (64) two-dimensional configurations is computed by the function $\text{AND_Match}(DD1DD0DD1DDDDD, X)$.

A set of functions such as these may be used to compute an approximation of the convex hull by detecting various local concavities, filling them in with 1's, and repeating the process until no further concavities are detected.

In some subsequent sections, operations involving interlevel interaction are illustrated. A pattern often used for such operations is

$$P_{\text{sons}} = DDDDDDDDDD1111.$$

For notational convenience, functions based upon the matching instructions will usually be written as functions of a single bit-pyramid parameter. An example is the following:

$$\text{ANDUP}(X) = \text{AND_Match}(P_{\text{sons}})(X) = \text{AND_Match}(P_{\text{sons}}, X).$$

2.4 Instruction Set

The instruction set for a pyramid machine can usually be divided into two parts: operations of the processing elements in the hierarchical domain, and operations of the controller module. While we could easily assume that the controller module is a standard Von-Neumann-style computer, it is generally unnecessary to assume that. Most important for the controller is the ability to test certain conditions that may exist in the pyramidal array and to jump conditionally. Also important is the ability to call and return from subroutines, and to load a base register used in computing the effective addresses for the PE's local memories. The architecture of pyramid machines is described in [Tanimoto 1981], [Dyer 1981], and [Tanimoto 1983a].

A minimal set of instructions for the PE's would include these:

1. LOAD bit pyramid into AC from PE local memories.
2. AND the bit pyramid in the AC with one in local memory, with the result going back into the AC.
3. NEGATE the contents of the AC.
4. STORE the contents of the AC into local memory.
5. COPY the value of the AC of neighbor number m into the AC.

However, we shall assume that the following PE instructions are also available:

1. CLEAR AC (set the accumulator value to 0).
2. SET AC to contain 1's at every cell.
3. LOAD C (move a bit pyramid into the mask register from PE local memories).
4. CLEAR C (reset the mask register).
5. OR the bit pyramid in the AC with one in local memory, with the result going back into the AC.
6. AND_Match: a pattern is specified (as described previously) and if all pattern elements match their corresponding neighborhood values at a cell, a 1 is written in the AC for that cell; otherwise a 0 is put in the AC.
7. OR_Match: a pattern is specified (as described previously) and if any pattern elements match their corresponding neighborhood values at a cell, a 1 is written in the AC for that cell; otherwise a 0 is put in the AC.

This model for a pyramid machine is the one which the author has found to be a useful compromise between the simple structures that make hardware implementation easy and the more complex structures that offer features useful in particular algorithms and approaches. One of the alternatives to that presented here is the "overlapped pyramid" model of Rosenfeld and his associates [Narayanan et al 1980], [Burt et al 1981], [Hong et al 1982], [Cibulskis and Dyer 1984].

3 Software Tools

Algorithms for pyramid machines span a range of complexities from very simple ones, implementable with short sequences of instructions, to highly-complicated algorithms involving data-dependent control, and many pyramids of data representing a variety of kinds of information. While the simple algorithms can often be developed with only pencil and paper, the complicated ones require a powerful algorithm-development environment to support their conception, expression, debugging, and evaluation. Various software tools contribute to such an environment. They include pyramid-machine emulators, graphical programming tools, and graphical monitoring tools.

3.1 Virtual Machines and Emulators

For purposes of algorithm development, it is often helpful to specify virtual pyramid machines that operate with higher-level instructions and larger data structures than might be actually used at the lowest level of machine language. While it is often instructive to program pyramid machines in machine language, one would not get very far without escaping from this level.

If the algorithm-development environment does not include an actual pyramid machine, the design of a simulator is an important issue. One may design a simulator at a low level which, in each emulated cycle, examines the current state of every simulated PE and updates it in accordance with the operation being performed. Such a simulation is useful in verifying that an algorithm is accurrately mapped onto the pyramid-machine architecture and in estimating the running time on the target machine. However, such a low-level simulation is usually very slow, and thus it does not permit the rapid feedback for algorithm development that is desirable. A more effective solution is to program intermediate-level operations (such as the thresholding of an image, the computation of a 3 by 3 convolution, the filling of a region using a pyramid-fill method, etc.) so as to run efficiently on the machine for simulation. Then, complex pyramid-machine algorithms can be constructed with calls to these intermediate-level routines. In an ideal development environment, the level of simulation is dynamically flexible. At one moment, segmentation routines may be running at high speed, with no attempt to do things the way they would be

done in a pyramid (except that the results, including side effects, must be the same as they would be after a real pyramid did the work); at the next moment, the individual bit pyramids representing regions in the process of segmentation might be shown undergoing an AND_Match operation [Blanford and Tanimoto 1986a].

3.2 Byte-Level Instruction Sets

Since the machine language of a pyramid machine may not support numerical operations directly, an improvement in programmability is gained by creating a byte-level (or higher-level) instruction set. The virtual architecture to support byte-level instructions provides a virtual 8-bit-wide accumulator, and it allows memory to be referenced in terms of byte pyramids. Such an instruction set includes operations such as the following:

1. LOAD byte pyramid from memory into the AC.
2. STORE byte pyramid from the AC into memory.
3. ADD byte pyramid in memory to the byte pyramid in the AC, with the result truncated (in case of overflow) and placed in the AC.
4. NEGATE the byte pyramid in the AC (take the two's complement of the AC).
5. MAXIMUM of the two values in the AC and in memory is put into the AC.
6. MINIMUM of the two values in the AC and in memory is put into the AC.

3.3 Higher-Level Instruction Sets

For certain kinds of applications, it may be useful to work with data types such as integer pyramids, "real pyramids" (pyramids of real numbers), and complex pyramids. It may also be of use to have special operations for symbol pyramids and set pyramids. Such operations can generally be effected (perhaps only approximately) with the basic pyramid machine. However, the provision of the high-level abstractions simplifies programming of such applications.

3.4 Design Aids: PyramidCalc

Graphical aids to algorithm development may usually be classified as either graphical aids to the design of program structure or graphical aids for monitoring or editing data objects.

The design of program structure involves the specification of the flow of control or the flow of data for the algorithm. Flow charts and data flow graphs

are the two principal classes of graphical forms for this aspect of algorithm design. This sort of graphical tool is not uniquely relevant to the design of pyramid-machine or even image-processing algorithms. Even the development of COBOL programs for payroll can potentially benefit from such tools.

On the other hand, graphical facilities for editing and monitoring the data structures used in image-processing systems are particularly appropriate to image-processing systems, rather than to, say, payroll-computing systems. Images themselves are inherently graphical, two-dimensional objects that may easily be displayed. Image data displayed as an image is generally easy for the programmer to understand, and it is very useful in algorithm development to display such data at various stages of the computation. Software tools that permit such displays to be easily generated, appropriately laid out on a graphics screen and effectively annotated, are very important. One such tool, called "PyramidCalc," has been developed for the Symbolics 3600 with color display [Blanford and Tanimoto 1985].

While providing good software tools for developing pyramid-machine programs is an important component of success in algorithm design, as a complement to the development, it is necessary to study the typical structures of pyramid algorithms themselves, and to understand the ways in which these structures may be combined to produce algorithms to solve new problems. To this end, we examine taxonomies for pyramid-machine algorithms.

4 Pyramid-Machine Algorithm Taxonomies

There are many different ways in which we might define a classification scheme for pyramid-machine algorithms. Here we consider some of them.

4.1 Data-Flow Patterns of Hanson and Riseman

In a multiprocessor system such as a pyramid machine, data must flow among the processors (unless they are all working independently). Such flow of data tends to follow certain general patterns, and one can usually describe an algorithm in terms of its pattern(s) of data flow.

Hanson and Riseman (1974) named three types of data flow in their "processing cone" architecture. Their terminology is applicable to pyramids in general and it provides an initial basis for discussion of pyramid data flow.

1. Reduction (Upward Flow) is a pattern of data movement from coarser to finer levels in a hierarchical domain.

2. Projection (Downward Flow) is a pattern of data movement from finer to coarser levels.

3. Lateral Flow refers to the movement of data horizontally; no data moves from one level to another during lateral flow, but stays in its current level.

4.2 The Data Structures Manipulated

One may classify algorithms on the basis of what data structures they work with. In the case of pyramid-machine algorithms, the following categories are often of use:

1. Bit Pyramids: the algorithm works with binary images and their pyramidal transforms, or it works with sets of cells in a hierarchical domain.
2. Byte Pyramids: the algorithm works with 8-bit approximations of real numbers, possibly representing gray-valued images.
3. Objects Located in Space: the algorithm works with objects, where an object is represented by some data in a PE's local memory. The coordinates of the PE are the coordinates of the object.

4.3 The Problem Being Solved

Another basis for classifying algorithms is according to the problem they solve or the transformation that they compute. Some examples are the following: edge detection, region growing, low-pass filtering (e.g., median filtering), and computing convex hulls.

4.4 Program Structure or Algorithm Paradigm

Perhaps the most interesting classification scheme for algorithms, from the point of view of designing pyramid-machine algorithms, is based on the control structure and the data-flow together. Each combination may be regarded as a paradigm, or general model for its class of algorithms. These paradigms may also be regarded as strategies for algorithm design [Tanimoto 1982]. Some paradigms for pyramid-machine algorithms are these: building pyramids, using pyramids as search trees, propagation, relaxation, emulating arbitrary interconnection networks, and parallel subpyramids. These paradigms are treated in more detail in the individual sections of this paper which follow.

5 Building Pyramids

Some of the simplest algorithms for pyramid machines are those that construct pyramid data structures of various kinds. These algorithms are very important in the design of more complex algorithms. Some types of pyramids include bit pyramids, real pyramids, and parameter-vector pyramids. Within each of these types there are subtypes of pyramids and algorithms to construct each subtype.

5.1 Bit Pyramids

The AND pyramid and OR pyramid may be regarded as the two most basic and important kinds of bit pyramids.

1. AND, OR, Count-of-2, Count-of-3. Count-of-k pyramids are analogous to AND pyramids and OR pyramids in that they may be defined in terms of summing the children and thresholding the sum to determine the home cell value. The AND pyramid uses a threshold of 4, the OR pyramid uses a threshold of 1, whereas values of 2 and 3 may also be used for building bit pyramids from a binary image [Tanimoto 1984].

2. Exclusive-Or and Coded Image Pyramids. Pyramids may be used to "factor" the common information out of blocks of an image so as to effect some data compression. For example, quadtree pyramids can be easily defined, and then an explicit quadtree can be obtained from the quadtree pyramid [Tanimoto 1983b].

5.2 Real Pyramids

1. Average, Sum, Maximum, Minimum, Median. An *average pyramid* is one computed from an image in the base as follows: each cell (except the base-level cells) computes the arithmetic mean of the values of its four children. When this process is complete, the root of the pyramid contains the arithmetic mean (average) for the entire image in the base. A median pyramid may be defined as the pyramid in which each non-base cell stores the median of its four children's values. Since a true median may not always exist for a set of four values, in practice one may take the 2nd largest child value (or, just as good if taken consistently, the 3rd largest child value). Real pyramids such as average pyramids and maximum pyramids are useful in a great many image processing algorithms. For example, both can be used in "bright-spot" selection algorithms described in the next section.

2. Sampling. Rather than perform arithmetic in building a pyramid, one may simply copy the value from (say) the northwest son of each cell to the cell. The resulting pyramid may be just as satisfactory for "planning"-style image processing algorithms as one that incorporates arithmetic or comparison operations. Such pyramids have also been used in "progressive-refinement" image transmission [Sloan and Tanimoto 1979].

5.3 Parameter-Vector Pyramids

The pyramids used in image analysis need not represent image brightness information. They may store derived characteristics of an image such as local texture, contrast, etc. The simplest way to construct such pyramids is to perform a local transformation in the base of the pyramid obtaining, say, an edge

image. Then a pyramid may be constructed from the resulting image using a method such as averaging or maximizing.

1. Mean and Standard Deviation. The values at each cell of a pyramid may be considered to be vectors of parameters rather than simply scalars. The pair (mean, standard deviation) of brightness values can be used to represent in summary fashion the characteristics of the square region of the image in the base, that corresponds to the particular pyramid cell.

2. Local Texture. Local texture information can be given in a wide variety of ways. One of these is to give an angle expressing the dominant direction of texture, together with a magnitude telling how strong the texture is.

5.4 Building and Re-Building

Some algorithms using pyramids construct a pyramid and then go through a sequence of gradual modifications in which the pyramid is successively rebuilt.

1. Tournament Sort on a Pyramid Machine. The well-known sorting algorithm based on tournaments can be implemented on a pyramid machine. Initially, a tournament is held in which a maximum pyramid is constructed. Later, the winner is removed from the base, and a runner-up is found. Gradually, the total ordering of the base-level values is made explicit [Tanimoto 1983c].

2. Parity Finding with Iteration Gives Bit Counting. Another algorithm that builds and rebuilds a pyramid is used for counting the number of 1's in the base of a pyramid. This algorithm has been described in the literature by several authors including [Rosenfeld 1979].

6 Using Pyramids as Search Trees

A simple, but very useful, paradigm is the "search tree" paradigm. By regarding a pyramid data structure as a search tree, providing steering information into an image, algorithms may rapidly retrieve the coordinates and values stored at certain "interesting" cells in the base of the hierarchical domain.

6.1 Using OR Pyramids to Locate 1s in the Base

In a sparse binary image, it may take a conventional computer a long time to find the first cell containing a 1. However, in a pyramid machine, a bottom-up pass followed by a top-down pass can effect the job in $O(\log n)$ steps. This can be useful in "iconic indexing" [Tanimoto 1976]. Here pictorial (spatial aspects of the image) attributes are used to locate points which may have symbolic information associated with them directly.

6.2 Bright-Spot Detection Methods

Using the paradigm that starts with a bottom-up pass and then performs a top-down pass, Certain "interesting" points of an image can be located in time proportional to the logarithm of the size of the image. These "bright-spot" techniques offer a surprizing degree of control, and they are useful in such areas as "verification vision" and seeding of region-growing techniques [Blanford and Tanimoto 1986b].

7 Propagation in Pyramids

Unlike building pyramids and using them as search trees, in which data moves only vertically in the pyramid, propagation permits horiontal movement and sometimes uses both horiontal and vertical movement simultaneously. Information moves along wavefronts, outwards from seed points or seed regions.

The idea of using "propagation" as a general approach to programming mesh-connected parallel computers is due to Duff (1980). This approach allows the programmer to design image operations that have global or semi-global effects, but that are specified largely in terms of local transformations in the image.

An important application of propagation is in the rapid extraction of the connected components of a binary image. Propagation through hierarchical domains can be faster than that through flat arrays, because the upper levels can provide shortcuts across large areas of the image. The "Pyramid-fill" algorithm is based upon this observation.

7.1 The Model

Propagation can be formulated in a general way. The model uses the notion of a "propagation condition" and a "propagation action." The general programmatic formulation of propagation is as follows:

```
UNTIL NO CHANGE IN THE MARK PYRAMID DO BEGIN
     AT EACH CELL DO BEGIN
          IF THE PROPAGATION CONDITION IS TRUE
            THEN PERFORM THE PROPAGATION ACTION
     END
END
```

Here the MARK pyramid is a bit pyramid that is used to prevent oscillatory behavior. Part of the propagation action is to set the cell's mark pyramid value to 1, thus "marking" the cell as having already participated in the propagation.

The propagation condition can often be conveniently decomposed into two subconditions, the "activation condition" and the "limitation condition." The formulation of propagation is then as follows:

```
UNTIL NO CHANGE IN THE MARK PYRAMID DO BEGIN
     AT EACH CELL DO BEGIN
          IF THE ACTIVATION CONDITION IS TRUE AND
             THE LIMITATION CONDITION IS NOT TRUE
            THEN PERFORM THE PROPAGATION ACTION
     END
END
```

The activation condition indicates how a PE is to be "awakened" while the limitation condition restricts the set of PE's which are permitted to respond to the awakening condition.

7.2 Application to Region Extraction

We now give an example of an algorithm based upon the propagation paradigm. This algorithm is called the *Region-Extraction Algorithm.* The input to the algorithm is an AND pyramid A whose base contains a set of regions which are the separate connected components of a binary image. Also input is a "seed image" which is a bit pyramid B in which all cells are 0, except that one cell in the base has value 1; that cell must be one which has value 1 in the input pyramid A. The algorithm is specified by describing the activation condition, limitation condition and propagation action. We let the mark pyramid be B. The activation condition is true at a cell if any of its neighbors are marked. The limitation condition is true at a cell if the value of pyramid A at that cell is 0. The propagation action is simply to mark the cell (i.e., write a 1 into B).

The loop must eventually terminate because there are only a finite number of cells that can ever be marked. As soon as an iteration is executed during which no cell gets marked, the "no-change" condition becomes true and the loop terminates.

8 Relaxation in Pyramids

8.1 Motivation

Although propagation permits information flow in all directions within a hierarchical domain, the information is constrained to move away from seed points and seed regions. Relaxation is more permissive, as information may flow in both directions along each path.

Relaxation, being somewhat more general than propagation, has less structure in its formulation. A consequence of this is that it is easy to specify a relaxation program that does not converge. Attention must therefore be paid to ensuring that relaxation programs do actually converge or are otherwise prevented from running away.

8.2 The Model

Relaxation can be formulated as follows:

```
WHILE THE TERMINATION CONDITION IS NOT TRUE DO BEGIN
     PERFORM THE RELAXATION ACTION
END
```

As with propagation, it is common to use as the termination condition the attainment of stability; relaxation algorithms are usually designed so that they converge on a solution, and once they have converged, the state of the system no longer changes with any additional iterations. Designing such procedures, one must take care to eliminate the possibility of divergence. When the relaxation cannot be guaranteed to converge naturally, a maximum number of iterations may be programmed into the algorithm (although then, one may question whether the algorithm should still be called a relaxation algorithm).

The relaxation action may involve the adjustment of the values of continuous parameters (e.g., probability estimates) or the adjustment of discrete labels or sets of labels. In "pyramid linking" schemes, a pointer at each cell of the pyramid can be changed during an iteration of relaxation to indicate a change in preference for a cell-to-cell association.

8.3 Applications

Relaxation can be applied in a pyramid, as it is in meshes, to such tasks as edge linking and region growing. However, in a pyramid, it is possible to express conditions for consistency which take somewhat more global information into account. For example, the label on a cell can be strongly influenced by the label on its parent or the labels on its children. Furthermore, it is possible to design relaxation procedures that place more weight on the influences of certain pyramid levels than on others, so that the most important spatial frequency components of an image have the greatest say in the resulting image description.

Pyramid-linking schemes have been devised (e.g., see [Burt et al 1981], [Hong 1982]) that use the relaxation paradigm to achieve segmentations of gray-level images into regions.

9 Emulating Arbitrary Interconnections

One view of a pyramid machine is as a set of processing elements with an elaborate interconnection network. A variation of this view is that the upper levels of the pyramid form an extra interconnection network for the processing elements in the mesh at the base. Overall, the pyramidal interconnection network is sufficiently general that one could use it to emulate any other interconnection network. For example, one could map a hypercube onto a pyramid using

two-by-two blocks of processing elements in the base, and then simulating the hypercube's links along certain dimensions by using the vertical connections of the pyramid.

It is has been shown by Miller and Stout how the upper pyramid levels may be used to increase the bandwidth of arbitrary data transfers among the processing elements in the base of the pyramid [Miller 1984].

10 Subpyramids in Parallel

One way to use the parallelism in a pyramid machine is to ignore the uppermost few levels of the pyramid, and consider the remaining levels (which form a truncated pyramid) as a mesh of smaller pyramids. Each of the smaller pyramids can be used to process a "window" of the image, so that a large set of windows are processed at the same time. The program to process each window may use a sequential format. For example, order statistics may be computed in each window using a "tournament-sort" approach. This approach may be used to effect median filtering of an image [Tanimoto 1983c].

11 Discussion

A number of paradigms have been presented here for the design of algorithms for pyramid machines. Other paradigms exist such as the use of pyramids to achieve scale invariance [Rosenfeld 1983], the bottom-up computation of curvature [Hartley and Rosenfeld 1985], and the parallel computation of separate-resolution components of a solution followed by the combining of those components. The latter paradigm is not yet as well understood as those that have been discussed here.

One of the attractive qualities of pyramid machines is that they offer so many different ways of programming. The flexibility seems likely to make pyramids very useful in solving complex machine vision problems that require fast algorithms.

12 References

1. Batcher, K. E. 1980. Design of a massively parallel processor. *IEEE Trans. Computers*, Vol. *C*-29, pp.836-840.

2. Blanford, R. P. and Tanimoto, S. L. 1985. The PyramidCalc system for research in pyramid machine algorithms. Technical Report 85-10-02, Dept. of Computer Science, University of Washington, October (also, to be presented at the 2nd International Workshop on Visual Languages, Dallas TX, June 1986).

3. Blanford, R. P. and Tanimoto, S. L. 1986a. A pyramid machine simulator for the Symbolics 3600. (to be presented at CVPR 86, Miami Beach, FL, June 1986).

4. Blanford, R. P. and Tanimoto, S. L. 1986b. Bright-spot detection in pyramids. (to be presented at the 8th International Conf. on Pattern Recognition, Paris, France, October).

5. Burt, P., Hong, T. H., and Rosenfeld, A. 1981. Segmentation and estimation of image region properties through cooperative hierarchical computation. *IEEE Transactions on Systems, Man, and Cybernetics*, Vol SMC-11, pp.802-804.

6. Cantoni, V., Ferretti, M., Levialdi, S., and Maloberti, F. 1985. A pyramid project using integrated technology. In Levialdi, S. (ed.), *Integrated Technology for Parallel Image Processing.* London: Academic Press, pp.121-132.

7. Cibulskis, J. M. and Dyer, C. R. 1984. An analysis of node linking in overlapped pyramids. *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC-14, No. 3, pp.424-436.

8. Delp, E. J., Siegel, H. J., Whinston, A., and Jamieson, L. H. 1985 An intelligent operating system for executing image understanding tasks on a reconfigurable parallel architecture. *Proc. 1985 IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Image Database Management*, Miami Beach, FL, November 18-20, pp.217-224.

9. Duff, M. J. B. 1976. CLIP4: a large scale integrated circuit array parallel processor. *Proc. Third International Joint Conf. on Pattern Recognition*, pp.728-733.

10. Duff, M. J. B. 1978. A review of the CLIP image processing system. *Proceedings of the National Computer Conference*, pp.1055-1060.

11. Duff, M. J. B. 1980. Propagation in cellular logic arrays. *Proc. Workshop on Picture Data Description and Management*, Asilomar Conf. Grounds, Pacific Grove, CA. IEEE Computer Soc. August, 1980, pp.259-262.

12. Dyer, C. R. 1981. A VLSI pyramid machine for hierarchical parallel image processing. *Proc. PRIP '81*, Dallas TX, August, pp.381-386.

13. Glinert, E. P. and Tanimoto, S. L. 1984. Pict: An interactive graphical programming environment. *IEEE Computer*, Vol. 17, No. 11 (November), pp.7-25.

14. Gottlieb, A., Grishman, R., Kruskal, C. P., McAuliffe, K. P., Rudolph, L., and Snir, M. 1983, The NYU Ultracomputer – designing an MIMD shared memory parallel computer. *IEEE Trans. Computers*, Vol. *C*-32, pp.175-189.

15. Hanson, A. and Riseman, E. M. 1974. Design of a semantically-directed vision processor. COINS-74-1, University of Massachusetts, Amherst, MA.

16. Hartley, R., and Rosenfeld, A. 1985. Hierarchical line linking for corner detection. In Levialdi, S. (ed.), *Parallel Integrated Technology for Image Processing*, London: Academic Press, pp.101-119.

17. Hong, T. H. 1982. *Pyramid Methods in Image Analysis.* Ph.D. Thesis, University of Maryland, College Park, Maryland.

18. Hong, T. H., Shneier, M. O., and Rosenfeld, A. 1982. Border extraction using linked edge pyramids. *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC-12, No. 5, pp.660-668.

19. Jarvis, J. F. 1982. Research directions in industrial machine vision. *IEEE Computer*, Vol. 15, No. 12, December, pp.55-61.

20. Kent, E. W., Shneier, M. O., and Lumia, R. L. 1985. PIPE (Pipelined image processing engine). *Journal of Parallel and Distributed Computing*, Vol. 2, pp.50-78.

21. Kent, E. W. and Tanimoto, S. L. 1985. Hierarchical cellular logic and the PIPE processor: Structural and functional correspondence. *Proc. 1985 IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Image Database Management*, Miami Beach, FL, November 18-20, pp.311-319.

22. Kuehn, J. T., Siegel, H. J., Tuomenoksa, D. L., and Adams, G. B., III. 1985. The use and design of PASM. In Levialdi, S. (ed.), *Parallel Integrated Technology for Image Processing*, London: Academic Press, pp.133-152.

23. Li, Z. and Uhr, L. 1985. Comparative timings for a neuron-recognition program on serial and pyramid computers. *Proc. 1985 IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Image Database Management*, Miami Beach, FL, November 18-20, pp.99-106.

24. Miller, R. 1984. *Pyramid Machine Algorithms for Image Processing.* Ph.D. Dissertation, Dept. of Mathematical Sciences, State Univ. of New York at Binghamton, Binghamton, NY.

25. Mudge, T. N. 1985. Vision algorithms for hypercube machines. *Proc. 1985 IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Image Database Management*, Miami Beach, FL, November 18-20, pp.225-320.

26. Narayanan, K. A., Peleg, S., Rosenfeld, A., and Silberberg, T. 1980. Iterative image smoothing and segmentation by weighted pyramid linking. Technical Report TR-989, Computer Science Center, University of Maryland, College Park, MD.

27. Pfeiffer, J. J., Jr. 1985, unpublished presentation at an informal workshop on pyramidal image processing systems, held at New Mexico State University, Las Cruces, NM, April 12-13, 1985. Essential features also communicated in: Integrating high and low level computer vision. *Proc. 1985 IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Image Database Management*, Miami Beach, FL, November 18-20, pp.119-125.

28. Preston, K., Jr., Duff, M. J. B., Levialdi, S., Norgren, P. E., and Toriwaki, J.-I. 1979. Basics of cellular logic with some applications in medical image processing. *Proceedings of the IEEE*, Vol. 67, No. 5, pp.826-856.

29. Reeves, A. 1985. Multicluster: An MIMD system for computer vision. In Levialdi, S. (ed.), *Parallel Integrated Technology for Image Processing*, London: Academic Press, pp.38-56.

30. Rosenfeld, A. (ed.) 1979. *Picture Languages: Formal Models for Picture Recognition.* New York: Academic Press.

31. Rosenfeld, A. 1983. Pyramids: Multiresolution image analysis. *Proc. Third Scandinavian Conference on Image Analysis*, Copenhagen, Denmark, July 1983, pp.23-28.

32. Rosenfeld, A. (ed.) 1984. *Multiresolution Image Processing and Analysis.* New York: Springer-Verlag.

33. Schaefer, D. H., Wilcox, G. C., Harris, V. J. 1985. A pyramid of MPP processing elements – experiences and plans. *Proc. 18th Annual Hawaii International Conference on System Sciences*, Vol. 1, pp.178-184.

34. Schneck, P. et al, 1985. Parallel processing programs in the federal government. *IEEE Computer*, Vol. 18, No. 16, June, pp.43-56.

35. Sharma, M., Patel, J. H., and Ahuja, N. 1985. NETRA: An architecture for a large-scale multiprocessor vision system. *Proc. 1985 IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Image Database Management*, Miami Beach, FL, November 18-20, pp.92-98.

36. Siegel, H. J. 1985. The PASM system and parallel image processing. In Freeman, H. and Pieroni, G. G. (eds), *Computer Architectures for Spatially Distributed Data*, New York: Springer-Verlag. (to appear).

37. Sloan,K. R., Jr. and Tanimoto, S. L. Progressive refinement of raster images. *IEEE Trans. Comput.*, Vol. C-28, No. 11, Nov. 1979, pp. 871-874.

38. Stout, Q. F. 1985. Mesh and pyramid computers inspired by geometric algorithms. *Proceedings of the Workshop on Algorithm-Guided Parallel Architectures for Automatic Target Recognition*, held at Leesburg, Virginia, July 16-18, 1984, Giles, C. L. and Rosenfeld, A. (eds), Feb. 1985, pp293-315.

39. Strong, J. P. 1985. The Fourier transform on mesh-connected processing arrays such as the Massively Parallel Processor. *Proc. 1985 IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Image Database Management*, Miami Beach, FL, November 18-20, pp.190-196.

40. Tanimoto, S. L. 1976. An iconic/symbolic data structuring scheme, *Proc. Joint Conf. on Pattern Recognition and Artificial Intelligence*, Hyannis, MA, June 1-3, 1976. Also in Chen, C. H. (ed), *Pattern Recognition and Artificial Intelligence*, New York: Academic Press, pp.452-471.

41. Tanimoto, S. L. 1981. Towards hierarchical cellular logic: Design considerations for pyramid machines. Dept. of Comp. Sci. Technical Report 81-02-01, February 1981.

42. Tanimoto, S. L. 1982. Programming techniques for hierarchical parallel processors. In Uhr, L. (ed.), *Multicomputers and Image Processing: Algorithms and Programs*, NY: Academic Press, pp.421-429.

43. Tanimoto, S. L. 1983a. A pyramidal approach to parallel processing. *Proceedings of the 10th International Symposium on Computer Architecture*, held at Stockholm, Sweden, June, 1983, pp.372-378.

44. Tanimoto, S. L. 1983b. Cellular logic operations in a hierarchical framework. *Proc. Third Scandinavian Conference on Image Analysis*, Copenhagen, Denmark, July 1983, pp.237-243.

45. Tanimoto, S. L. 1983c. Algorithms for median filtering of images on a pyramid machine. Chapter Nine of *Computing Structures for Image Processing*, M.J.B. Duff (ed.) Academic Press, 1983, pp.123-141.

46. Tanimoto, S. L. 1984. A hierarchical cellular logic for pyramid computers. *Journal of Parallel and Distributed Computing*, Vol. 1, No. 2, pp.105-132.

47. Tanimoto, S. L. 1985. An approach to the iconic/symbolic interface. In Levialdi, S. (ed.), *Parallel Integrated Technology for Image Processing*, London: Academic Press, pp.31-37.

48. Tanimoto, S. L., Crettez, J.-P., and Simon, J.-C. 1984. Alternative hierarchies for cellular logic. *Proc. 7th International Conf. on Pattern Recognition*, Montreal, Canada, July-August, pp.236-239.

49. Tanimoto, S. L., Ligocki, T. J., and Ling, R. 1986. A prototype pyramid for hierarchical cellular logic (tentative title). To appear in L. Uhr (ed.) *Massively Parallel Multicomputers for Vision* (also tentative title).

50. Tanimoto, S. L., and Pavlidis, T. 1975. A hierarchical data structure for picture processing. *Computer Graphics and Image Processing*, Vol. 4, pp.104-119.

51. Uhr, L. 1984. *Algorithm-Structured Computer Arrays and Networks: Architectures and Processes for Images, Percepts, Models, Information*, New York: Academic Press.

PYRAMID ALGORITHMS ON PROCESSOR ARRAYS

Anthony P. Reeves
School of Electrical Engineering
Cornell University
Ithaca, New York 14853

Abstract

A class of adaptive grid size algorithms, called pyramid algorithms, have received much attention in recent years for computer vision applications. Special hardware architectures which are optimal for grid resolution transformations have been proposed to implement these algorithms. In this chapter analysis techniques are described which measure the effectiveness of different architectures. A comparison is made between the more conventional planar or flat architecture and the optimal pyramid architecture. It is shown that in many cases the additional hardware of the pyramid scheme offers little improvement in performance.

INTRODUCTION

The general concept of pyramid algorithms is to decompose a large matrix into a number of lower resolution matrices for more convenient processing. The most typical pyramid structure consists of a square base matrix with dimensions N x N (where N is a power of 2) and $\log_2 N$ reduced matrices (layers) each layer having dimensions half that of the layer below until a single element apex (layer 0) is reached. A number of algorithms for image processing have been based on this data structure. Other applications such as multigrid techniques for PDE's can also be considered in the pyramid framework.

In order to achieve the very high data processing rates needed for image processing applications a number of highly parallel computer architectures have been designed and implemented; several of such systems are now commercially available. A very effective parallel computer architecture for conventional image processing applications is the mesh connected processor array. For processing pyramid data structures a number of researchers have proposed hardware modifications to the two dimensional processor array. These modifications directly match the pyramid data structure by having interconnected layers of processor arrays attached to the base processor array. In this paper techniques are developed to investigate the efficiency with which pyramid algorithms can be implemented on processor arrays and the effectiveness of enhanced hardware schemes is considered.

NATO ASI Series, Vol. F 25
Pyramidal Systems for Computer Vision
Edited by V. Cantoni and S. Levialdi
© Springer-Verlag Berlin Heidelberg 1986

A major consideration in the design of a highly parallel computer architecture is the processor interconnection network. Processors must communicate with a speed which does not impede data processing; however, a general processor interconnection network is usually prohibitively expensive when a large number of processors are involved. A major design task is to design a restricted network which is adequate for the anticipated tasks for the system. The mesh interconnection scheme is simple to implement and is well suited to a large number of image processing algorithms. The first section of this paper describes the Massively Parallel Processor (MPP) [1] and the performance of the processor interconnection network is considered in detail.

The second section of this paper deals with the implementation of pyramid algorithms on the MPP and also considers the effect of adding a pyramid hardware structure to the MPP. A two dimensional processor array will be called a *flat* processor and a processor array with additional hardware for pyramid data structures will be called a *pyramid* processor. The general techniques used to analyze these architectures may be used to determine the effectiveness of other highly parallel processor systems for pyramid applications.

The hardware enhancement of a pyramid of processing arrays approaches $\frac{4}{3}$ times the number of processing elements (PE's) required for the base array, as the number of layers increases. Therefore, the maximum possible increase in arithmetic capability is $\frac{4}{3}$. However, the major advantage claimed for the pyramid processor is the speedup in interprocessor data routing. In order to compute a global function over an N x N processor array N processing steps are required. However, on a pyramid processor global information may be extracted at the apex of the pyramid after $\log_2 N$ steps. The potential gain for the pyramid system is therefore $N/\log_2 N$; this value is shown in Table 1. for different values of N. The MPP has a 128 x128 base array therefore the potential speedup is in the order of 18.

The pyramid computer has an improved performance for two types of operations: (a) multiresolution algorithms which involve frequent changes between matrix resolutions and (b) global feature extraction operations. However, the additional pyramid hardware must be justified by the multiresolution operations alone since there are more cost effective hardware enhancements for global feature extraction [2].

Table 1: Data routing advantage of the Pyramid Architecture

N	4	8	16	32	64	128	256	512	1024
$N/\log_2 N$	2	2.33	4	6.4	10.66	18.29	32	56.8	102.4

The cost of the additional hardware for the pyramid processor has been considered by some to be simply $\frac{4}{3}$ times the cost of the flat processor since this is the increase in the number of PE's. The cost will, in general, be much higher than this since each PE now requires more interconnections (an increase from 4 to 9 in our example) and interprocessor connections are no longer local which could be particularly expensive for multiprocessor chip designs.

FLAT PROCESSOR ARRAYS

The Massively Parallel Processor consists of 16384 bit-serial Processing Elements (PE's) connected in 128 x 128 mesh [1]. That is each PE is connected to its 4 adjacent neighbors in a planar matrix. The two dimensional grid is one of the simplest interconnection topologies to implement, since the PE's themselves are set out in a planar grid fashion and all interconnections are between adjacent components. Furthermore, this topology is ideal for two dimensional filtering operations which are common to low level image processing such as small window convolution.

The PE's are bit-serial, i.e. the data paths are all one bit wide. This organization offers the maximum flexibility, at the expense of the highest degree of parallelism, with the minimum number of control lines. For example, as an alternative to the MPP consider 2048 8-bit wide PE's (on the MPP one chip contains 8 1-bit PE's). The 8-bit version would have a less rich set of instructions restricted to predefined byte operations while the bit-serial processors can process any data format. The advantage gained with the 8-bit system is that full processor utilization is achieved with arrays of 2048 elements while arrays of 16384 elements are required for full utilization of the MPP. The MPP PE is well matched to low level image processing tasks which often involve very large data arrays of short integers which may be from 1 to 16 bits.

The effectiveness of the MPP architecture for various interprocessor data manipulations is considered. The MPP offers a simple basic model for analysis since it involves just mesh interconnections and bit-serial PE's. The minimal architecture of the MPP is of particular interest to study, since any architecture modifications to improve performance would result in a more complex PE or a more dense interconnection strategy. The MPP is programmed in a high level language called Parallel Pascal [3].

The MPP Processing Element

The MPP processing element is shown in Fig. 1. All data paths are one bit wide and there are 8 PE's on a single CMOS chip with the local memory on external memory chips.

Except for the shift register, the design is essentially a minimal architecture of this type. The single bit full adder is used for arithmetic operations and the Boolean processor, which implements all 16 possible two input logical functions, is used for all other operations. The NN select unit is the interface to the interprocessor network and is used to select a value from one of the four adjacent PE's in the mesh.

The S register is used for I/O. A bit plane is slid into the S registers independent of the PE processing operation and it is then loaded into the local memory by cycle stealing one cycle. The G register is used in masked operations. When masking is enabled only PE's in which the G register is set perform any operations; the remainder are idle. The masked operation is a very common control feature in SIMD designs. Not shown in Fig. 1. is an OR bus

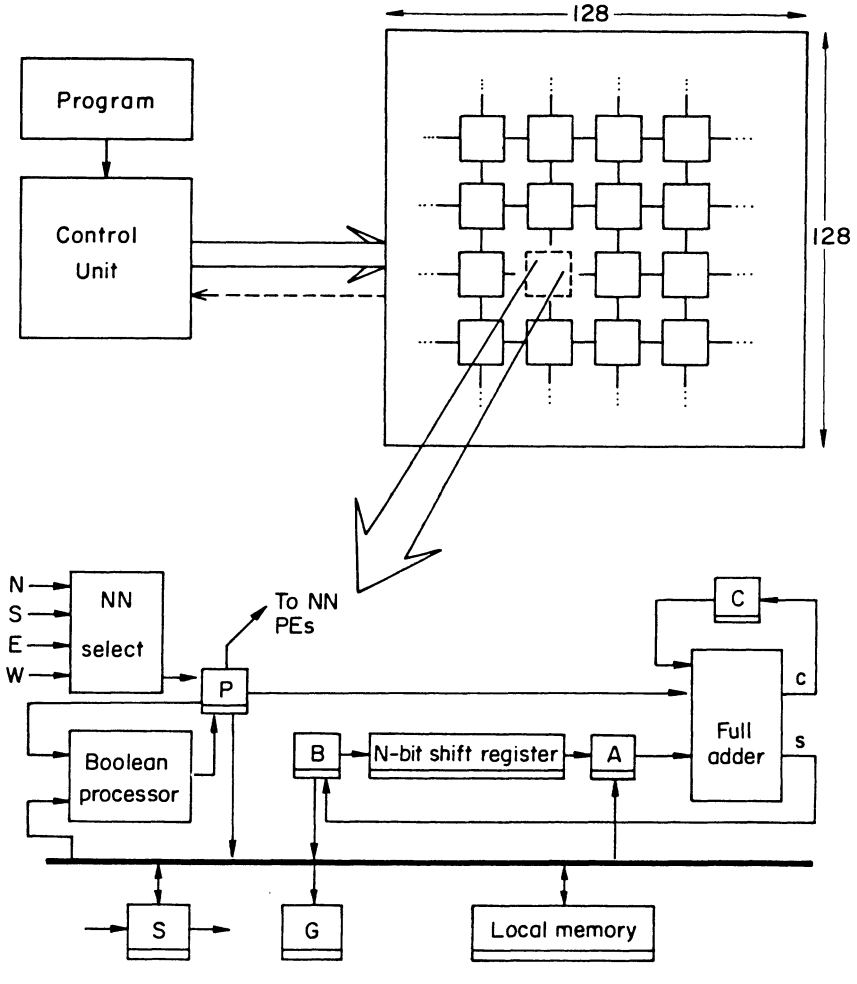


Figure 1. The MPP Processing Element

output from the PE. All these outputs are connected (ORed) together so that the control unit can determine if any bits are set in a bitplane in a single instruction. On the MPP the local memory has 1024 words (bits) and is implemented with bipolar chips which have a 35 ns access time.

The main novel feature of the MPP PE architecture is the reconfigurable shift register. It may be configured under program control to have a length from 2 to 30 bits. Improved performance is achieved by keeping operands circulating in the shift register which greatly reduces the number of local memory accesses and instructions. It speeds up integer multiplication by a factor of two and also has an important effect on floating-point performance.

Flat Array Performance Evaluation

In order to analyze the effectiveness of the interconnection network for different manipulations it is necessary to characterize the processing speed of the PE and the speed of the interconnection network. On the MPP both of these are data dependent; we have considered three representative cases: single-bit *Boolean* data, 8-bit *integer* data and 32-bit floating-point (*real*) data. For each of these data types we have estimated a *typical* time for an elemental operation. These estimates are of a reasonable order for this minimal PE architecture but are not very precise. For example, the instruction cycle time for a memory access and operation on the MPP is 100 ns. An elemental boolean operation may be considered to take 100 ns.; however, it may be argued that an operation should involve two operands and have all variables in memory in which case three memory accesses (instructions) would require 300ns. For our analysis a two instruction (200 ns) model was used to represent Boolean instruction times. For the real and integer data a convenient number midway between the times for addition and multiplication was used; this was 5 μs. for an integer operation and 40 μs. for a real operation. It should be remembered that elemental operations also include many other functions such as transcendental functions since these can be computed in times comparable to a multiplication on a bit-serial architecture. By adding a large amount of additional hardware to each PE it is possible to increase the speed of multiplication by 10 times or more [4].

For each of the data manipulations considered, times for the three different data types was computed. The performance of the MPP for each manipulation is indicated by the ratio of the data transfer time to an elemental PE operation on the same data type; this will be called the *transfer ratio*. One way to look at this ratio is the number of elemental data operations which must be performed between data transfers for the data transfers not to be the dominant cost for the algorithm. On the MPP data may be shifted between adjacent PE's in one instruction time (100 ns.) concurrently with a PE processing instruction.

For many applications the physical dimensions of the parallel hardware are smaller than the dimensions of the array to be processed. In this case the data array is processed as a set of blocks. An extension of the data manipulation algorithms to deal with this situation is discussed.

Shift and Rotate Operations

The only permutation function which is directly implemented by the MPP is the near neighbor rotate (or shift). The direction of the rotation may be in any of the four cardinal directions. The rotation utilizes the toroidal end around edge connections of the mesh. The *shift* function is similar except that the mesh is not toroidally connected and zeroes are shifted into elements at the edge of the array; therefore, the shift function is not a permutation function in the strict sense. The concept of the rotate and shift functions extends to n dimensions; on the MPP the last two dimensions of the array correspond to the parallel hardware dimensions and are executed in parallel, higher dimension operations are implemented in serial. The cost of the rotate function is dependent on the distance rotated. It also depends on the size of the data elements to be permuted.

The transfer ratios for the shift operation are given in Table 2. Ratios are given for shift distances of 1 and 64 elements; 64 is the largest shift which will normally be required in a single dimension on a 128 x 128 matrix since a shift of 65 can be obtained with a rotate of -63 and a mask operation. The worst case figures for a two dimensional shift is 64 in each direction; i.e., twice the figures given in Table 2.

For single element shifts the interconnection network is more than adequate for all data types. For maximum distance shifts the ratio of 33 for Boolean data could cause problems for some algorithms but the situation is much better for real data.

Large Array Processing

Frequently the data to be processed by a parallel processor will be in the format of arrays which exceed the fixed range of parallelism of the hardware. Therefore, it is

Table 2: The Cost for Shift and Rotate Operations

Shift distance	Operation cost in μs.			Transfer Ratio		
	Boolean	integer	real	Boolean	integer	real
1	0.2	1.6	6.4	1.0	0.32	0.16
64	6.5	51	210	33	10	5.2

necessary to have special algorithms that will deal with large arrays by breaking them down into blocks manageable by the hardware, without loosing track of the relationships between different blocks.

There are two main schemes for storing large arrays on processor arrays: the *blocked* scheme and the *crinkled* scheme. Consider that a M x M array is distributed on an N x N processor array where K = M / N is an integer. In the blocked scheme a large array is considered as a set of K x K blocks of size N x N each of which is distributed on the processor. Therefore, elements which ar allocated to a single PE are some multiple of N apart on the large array. In the crinkled scheme each PE contains a K x K matrix of adjacent elements of the large array. Therefore, each parallel processor array contains a sampled version of the large array. For conventional array operations which involve large array shift and rotate operations both blocked and crinkled schemes can be implemented with only a very small amount of overhead. The crinkled scheme is slightly more efficient when shift distances are very small and the blocked scheme has a slight advantage when the shift distance is of the order of N.

The Performance of the MPP

The transfer ratio for a number of important data manipulations is shown in Table 3. The figures for large arrays correspond to the blocked data scheme. For large arrays the transfer ratio is normalized by the cost of an operation on the whole array; i.e., K x K array operations.

This technique may be applied to almost any parallel architecture. It is expected that the results obtained for the MPP would be very similar to those obtained for other like architectures such as the Distributed Array Processor (DAP) [5] or NCR's GAPP processor chip which contains 72 PE's with local memory [6]. The results given in Table 3 could be used by programmers to predict the performance of algorithms on the MPP.

For the MPP, the results indicate that, although arbitrary data mappings may be very costly, some important data manipulations can be done very efficiently. The shift register, which has a 2 times speedup factor for multi-bit arithmetic also has a significant effect on the implementation of several of the multi-bit data manipulations studied. Especially interesting is the improvement of over 10 times for real data distribution. The shuffle cannot be implemented fast enough for efficient FFT implementation; however, other data mapping strategies for the FFT, such as butterfly permutations, are well known which have a much more efficient implementation on the MPP. A more detailed analysis of data mappings on the MPP is given in [7].

On the DAP row and column distribution is implemented directly by special hardware buses. For the MPP we can see from Table 3 that no advantage would be gained from this

hardware for real data operations and possibly very little advantage for integer operations.

PYRAMID PROCESSING

A Pyramid Architecture

A hypothetical architecture of a pyramid machine based on the MPP is illustrated in Fig. 2. The PE is similar to the MPP PE with the addition of five additional interprocessor connections. Four of these are connected to the four children at the next lower level and the fifth is connected to the parent PE in the layer above.

Pyramid Primitive Operations

We consider processing operations on pyramid data structures to be one of three types: *elemental*, in which no communication between PE's is involved, *horizontal*, in which adjacent elements in the same pyramid level are combined, and *vertical*, in which elements at different adjacent levels are combined. Furthermore, we consider that these operations may be applied to the total pyramid or to a subset of levels. Other relevant pyramid operations include feature extraction and data broadcast.

Pyramid Operations on a Flat Array

An efficient scheme for mapping a pyramid data structure to a N x N processor array is as follows. All levels of the pyramid which have dimensions less than N x N are mapped into a single plane. The level with size N x N is mapped into a second plane and levels lower than this are mapped in such a way that each PE is allocated to a sub-pyramid with

Table 3: Transfer Ratios for Different Data Manipulations and Array Sizes

Data Manipulation	Array Size								
	128 x 128			256 x 256			512 x 512		
	Boolean	integer	real	Boolean	integer	real	Boolean	integer	real
Data Shift									
a) 1 element	1.0	0.32	0.16	2.0	0.5	0.24	2.0	0.5	0.24
b) worst case	33	10.2	5.2	33	10	5.2	33	10	5.2
Broadcast									
a) Global	2	0.64	0.32	0.88	0.28	0.14	0.59	0.20	0.09
c) Row (or column)	68	3.2	0.52	35	1.68	0.30	18.2	0.92	0.19
Shuffle (2-dimensional)	640	90	42	640	90	42	640	90	42
Transpose	840	105	44	840	105	44	840	105	44

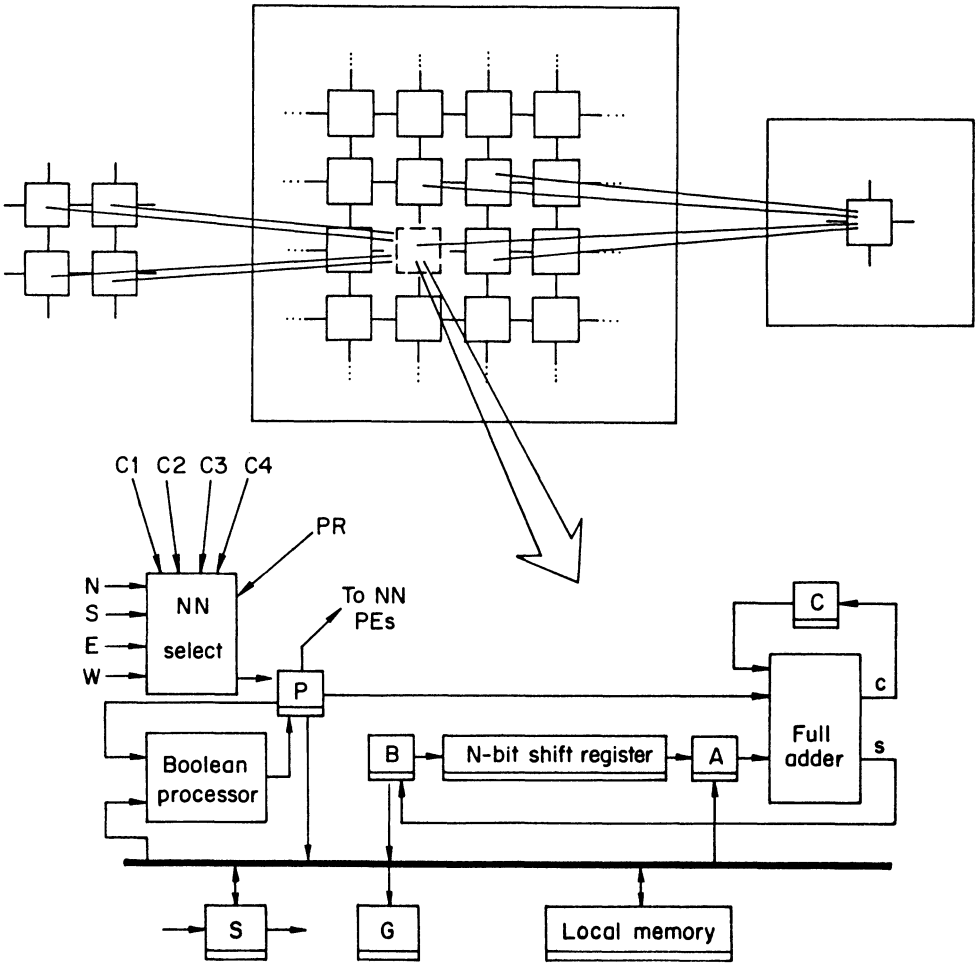


Figure 2. The Pyramid Processing Element

adjacent elements. This is achieved by storing each level with the crinkled storage format. Therefore, for a k-level pyramid the top n - 1 levels ($n = \log_2 N$) are stored in one plane, and the remaining k - n levels are stored in $\sum_{i=0}^{k-n-1} 2^{2i} = \frac{1}{3}(2^{2(k-n)}-1)$ planes. Data transitions between the top n - 1 levels is achieved with perfect shuffle permutations. The perfect shuffle for an 8 x 8 matrix is shown in Fig. 3. Each quadrant of the matrix is distributed over the whole matrix. A 4 x 4 pyramid structure embedded in an 8 x 8 matrix is shown in Fig. 4. With this organization, a transition to a lower level can be achieved by a shuffle and a transition to a higher level can be achieved with an inverse shuffle for all levels simultaneously. This scheme makes good use of memory and is optimal for horizontal operations but is

(a)

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

(b)

1	5	2	6	3	7	4	8
33	37	34	38	35	39	36	40
9	13	10	14	11	15	12	16
41	45	42	46	43	47	44	48
17	21	18	22	19	23	20	24
49	53	50	54	51	55	52	56
25	29	26	30	27	31	28	32
57	61	58	62	59	63	60	64

Figure 3. The Two Dimensional Perfect Shuffle.
The shuffled version of matrix (a) is shown in (b).

relatively slow for vertical operations.

A Comparative Analysis

To evaluate the cost of macro operations a *normalized cost* is used which is similar in concept to the transfer ratio. It is obtained by dividing the actual cost by the cost of an elemental operation on the data structure being processed by the macro operation.

Elemental Operations

The actual time cost in terms of the number of processor array arithmetic operations is shown in Fig. 5 for an elemental operation applied to the whole pyramid data structure and for an architecture with a 128 x 128 base size. The cost for a flat array is shown by the solid

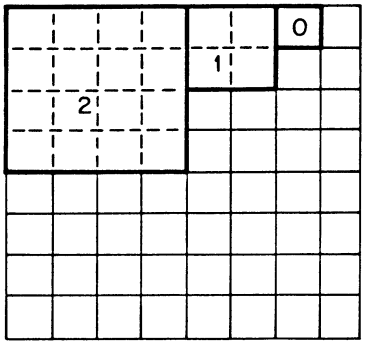


Figure 4. A 4 x 4 base pyramid embedded in an 8 x 8 mesh

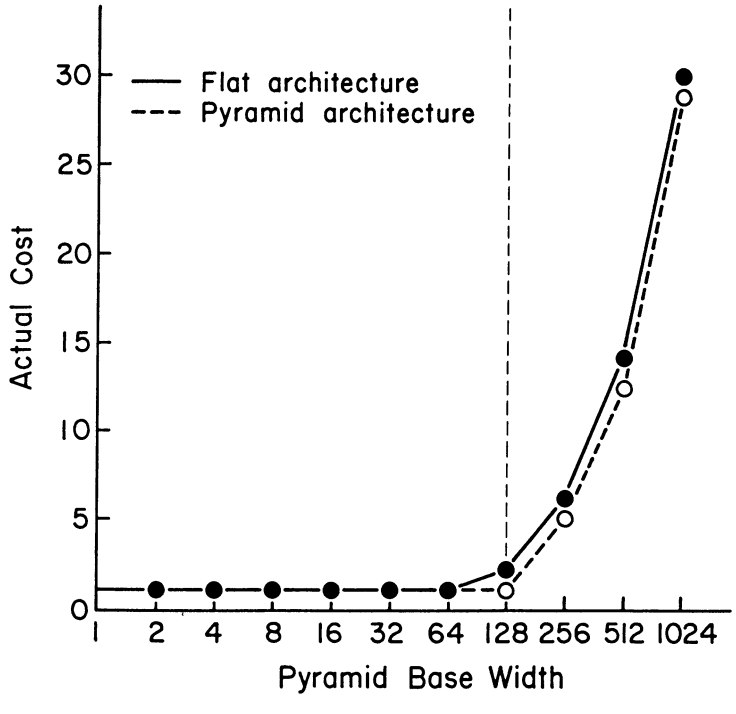


Figure 5. The Actual Cost for Elemental Pyramid Operations

line. While the base of the pyramid to be processed is less than 128 x 128 then the whole pyramid is embedded in a single array and only one operation is required. For a 128 x 128 base pyramid 2 operations are required and a blocked processing strategy is used for larger

pyramids. The cost for the pyramid architecture is shown by the broken line. In this case the 128 x 128 pyramid is exactly matched to the hardware and only requires one operation. Larger pyramids must be blocked using the pyramid base level.

The normalized cost of a elemental operation on the whole pyramid data structure is shown in Fig. 6. Since we normalize by the cost of an operation on the whole pyramid, in this case the result is a constant 1 for all pyramid sizes. For comparison purposes all normalizations are done with respect to an operation on a flat array. The broken line in Fig. 6 shows the normalized cost for an elemental operation for a true pyramid architecture with a 128 x 128 base. When the base of the pyramid to be processed is less than 128 then the cost is the same for both architectures. When the base is 128 x 128 then a 50% reduction in normalized cost for the pyramid architecture is observed. For pyramids with bases larger than 128 x 128 both architectures must process the data in blocks and the advantage of the pyramid architecture rapidly diminishes. It is important to note that the normalized cost for this graph and for all following graphs is plotted on a logarithmic scale.

Large Pyramid Processing

When the pyramid structure ro be processed has a base which is larger than the processor array then a blocking scheme must be used for the lower large levels of the pyramid. An effective scheme is to use the crinkled storage strategy for each large level. In this way each PE contains a complete sub pyramid; therefore, level changes for the large levels are done locally within PE's and no interprocessor communications are necessary. The crinkled

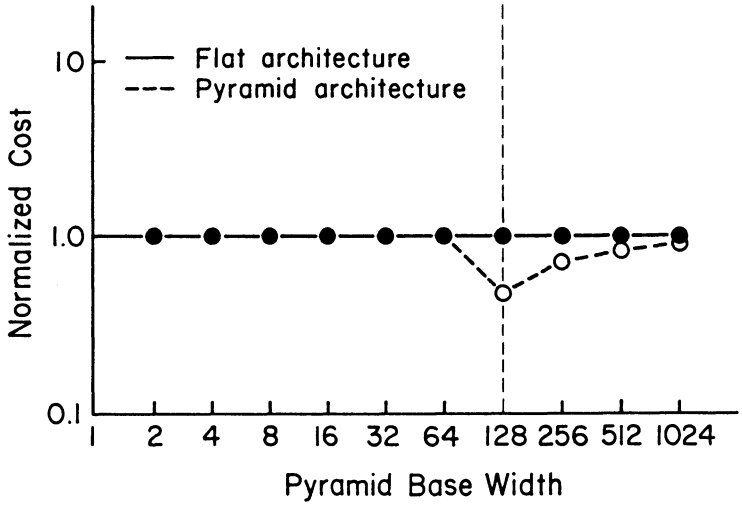


Figure 6. The Normalized Cost for Elemental Pyramid Operations

storage scheme is also very efficient for horizontal operations.

Horizontal Operations

Adjacent element information is readily available for both the architectures considered. Therefore, the normalized cost for macro operations involving horizontal near neighbor information will be similar to Fig. 6 except that the graph will be translated upwards by the time for the macro operation to be implemented in terms of elemental operations. The cost of some important horizontal operations is shown in Fig. 7. For reference, the cost of an add operation is first shown; it is less than one since an add requires less time than a multiply for integer and real data types. A 3 x 3 mean operation is achieved with four near neighbor additions, while a 3 x 3 convolution requires 9 multiplications and 8 near neighbor additions. A symmetric 5 x 5 convolution, such as that used by Burt for Pyramid building [8] requires 6 multiplications and 8 near neighbor additions. An example of an expensive macro operation is the 10 by 10 convolution which requires 100 multiplications and 99 near neighbor additions.

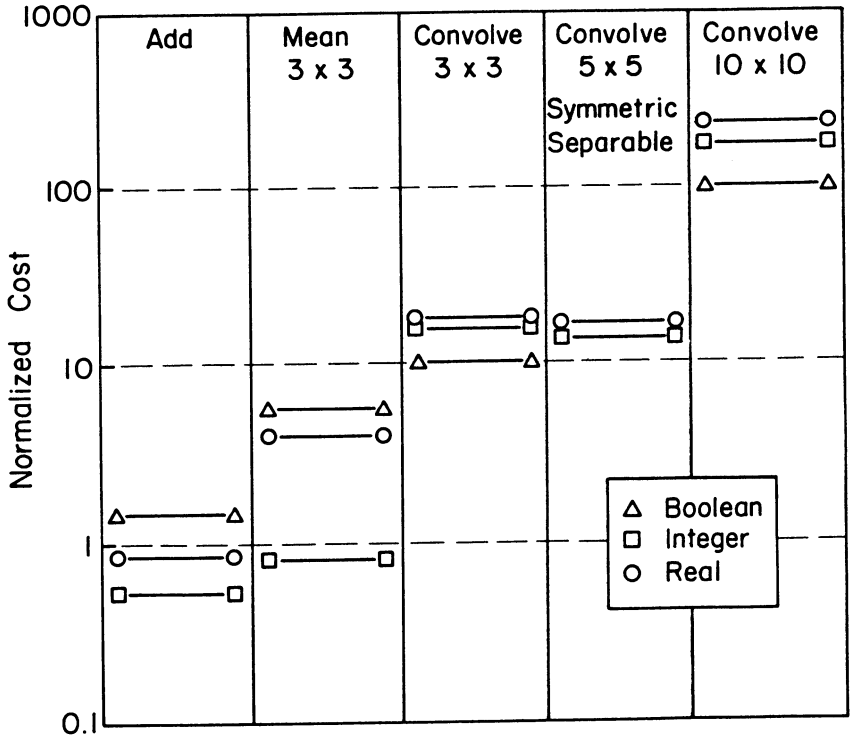


Figure 7. The Normalized Cost of some Horizontal Operations

Vertical Pyramid Operations

It is in the vertical operations that the pyramid architecture is expected to excel in comparison to a flat architecture. An important operation in many pyramid algorithms is to build a pyramid from an image located at the lowest level of the pyramid. We consider a very simple building operation in which each higher level is constructed from the mean of the four sons of the adjacent lower level.

Pyramid Building

The cost of pyramid building is shown for a flat processor in Fig. 8 and for a pyramid processor in Fig. 9. The building cost is normalized by an elemental operation applied to the whole pyramid data structure. We see that for the worst case base width of 128 the flat processor is several times slower that the pyramid however this advantage rapidly diminishes if large pyramids are processed. The Boolean case is particularly bad for the flat array

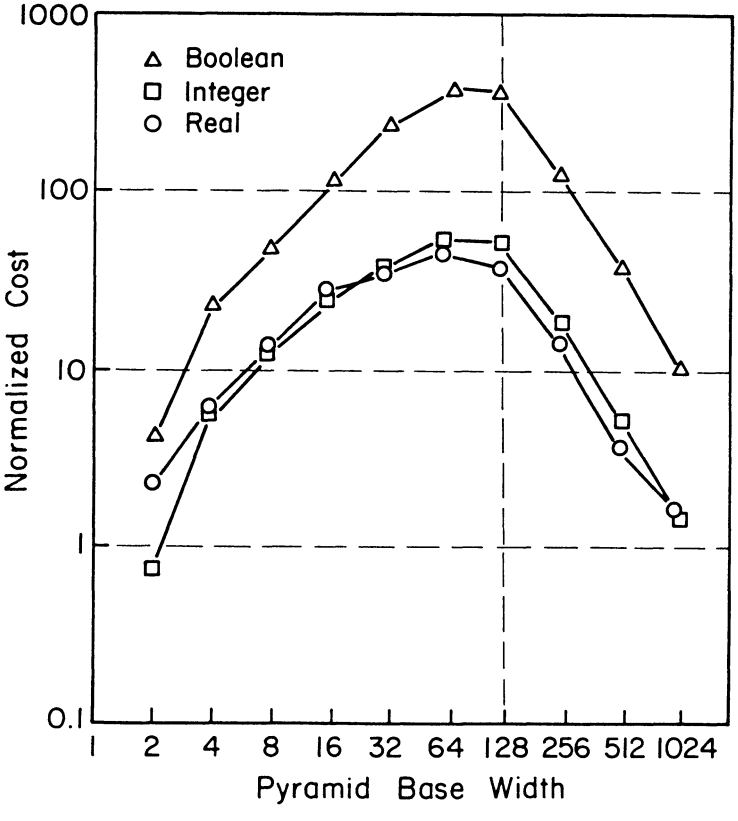


Figure 8. The Normalized Cost for Simple Pyramid Building on a Flat Processor Array

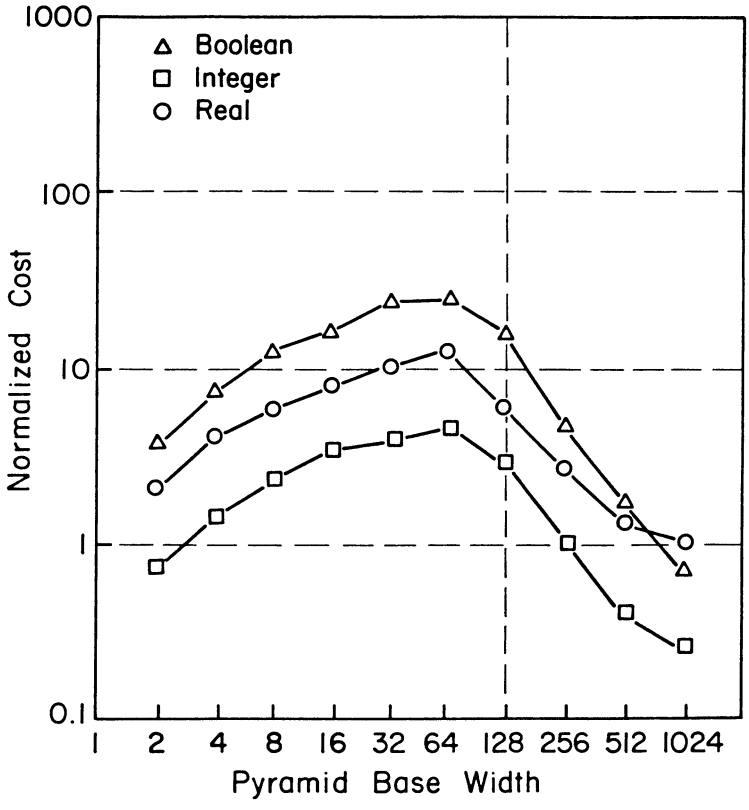


Figure 9. The Normalized Cost for Simple Pyramid Building on a Pyramid Processor Array

because the overhead to do the pyramid data manipulations cannot be amortized over the number of bits in the data as it can for the numeric data formats. The flat array although slower than the pyramid may not be significantly slower in a pyramid building algorithm. For example consider the overlapped algorithm of Burt [8]. At each level a 5 x 5 convolution must be computed before proceeding to the next. The cost of computing these convolutions is comparable to the data building cost.

Another way to look at the vertical operation cost is to consider a single level reduction; this is shown for both flat and pyramid processors in Fig. 10. In this case the cost is normalized by an elemental operation on the level at which the reduced data is generated. The architectures have the same performance when the base is smaller than the level being processed. The worst case for the flat processor occurs for the level which is half the base size since this requires the most interprocessor data transfers. If many operations fall on this level transition then one technique is to consider the processor array to have a size of 64 x 64. The arithmetic processing speed is reduced by four (since only a quarter of the processors are

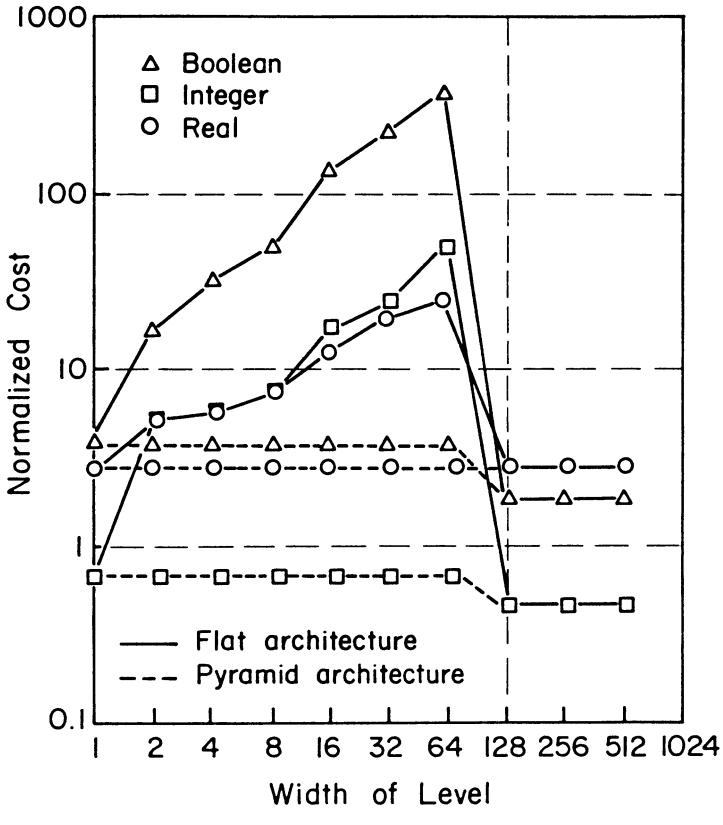


Figure 10. The Normalized Cost for Single Layer Reduction

used) but the data transition is managed at least ten times faster.

Data Reduction

In Fig. 11 and Fig. 12 the cost for computing the global sum of the elements in the base of the pyramid is given. In this case the cost is normalized by an elemental operation over the whole pyramid data structure. The advantage of the pyramid architecture is not as great as in the pyramid building example because the flat array is no longer forced to emulate the pyramid in the highest levels since only the final sum is required.

CONCLUSIONS

A general technique for determining the cost of pyramid algorithms has been presented. This technique may be used on any flat array architecture to evaluate its performance for pyramid algorithms and to evaluate the benefit of a pyramid processor hardware enhancement. Once the graphs similar to those given in Figs. 6-12 have been obtained then the mix

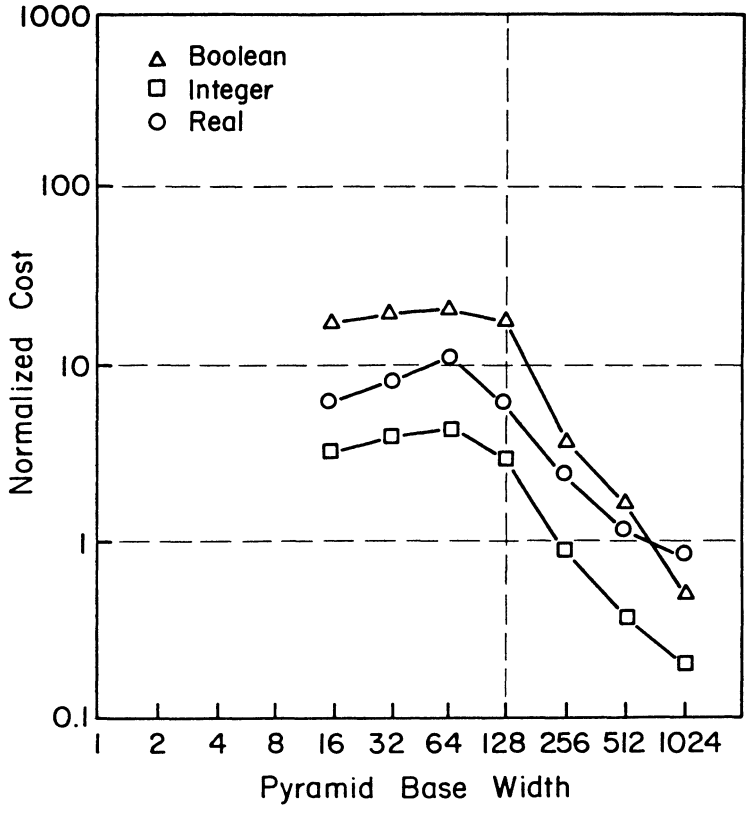


Figure 11. The Normalized Cost for Sum Reduction on a Flat Processor Array

of vertical to other pyramid operations for the desired application must also be determined for an accurate evaluation.

From the results given in this paper the following observations can be made.

1. Pyramid algorithms can be effectively implemented on a flat array in many cases; each application should be examined individually in detail.
2. If most of the operations are performed in the base of the pyramid then the pyramid hardware will offer little advantage.
3. The flat array is substantially worse for Boolean data types than for numeric data types. Therefore, a preponderance of Boolean data structures favors the pyramid approach. However, if numeric data types dominate then the case for pyramid hardware is much weaker.
4. The pyramid hardware can only be justified by the need for a large number of pyramid level changes in a algorithm. It cannot be justified by the global data reduction function alone since other techniques can be used on the flat array.

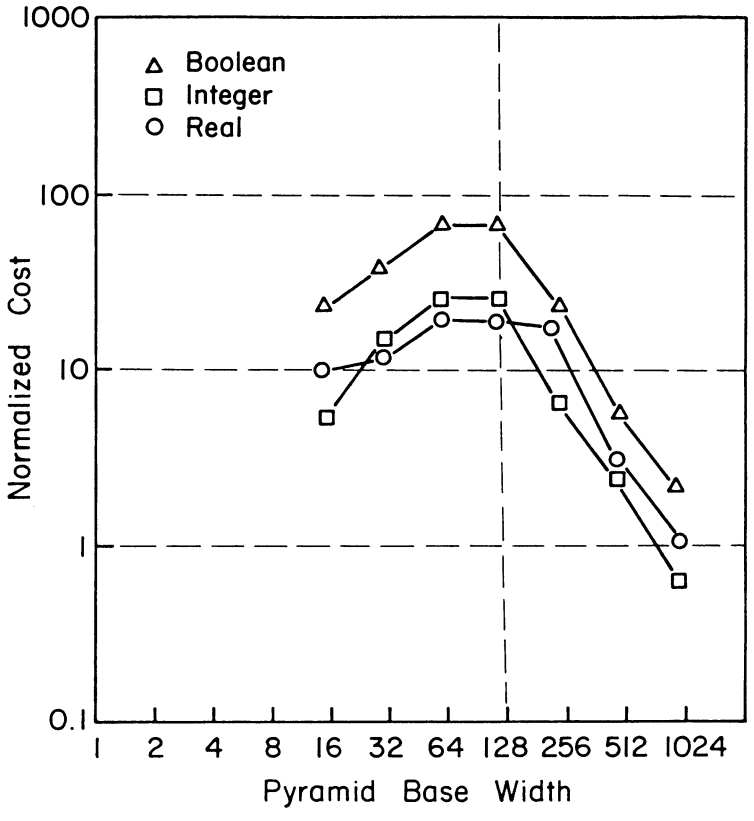


Figure 12. The Normalized Cost for Sum Reduction on a Pyramid Processor Array

5. The ratio of the hardware base size and the pyramid base size will have a major effect on the performance. For numeric data processing one way to increase the effectiveness of the flat array is to use a smaller number of multi-bit processors; therefore the hardware base size is reduced without loss of processing power.

6. When the pyramid to be processed is much larger than the hardware processor array then there is very little advantage in having the pyramid hardware. This is the case when the MPP is used to process pyramids with a base size of 512 x 512.

In summary while the analysis is complex and very application dependent there are a number of situations which may be identified for which justification for the pyramid architecture can be determined. The pyramid hardware is hard to justify if either (a) large pyramid processing techniques are used, (b) the operations to be performed are numeric, or (c) vertical operations do not constitute a large part of the pyramid algorithms to be implemented. The strongest case for the pyramid hardware exists when Boolean data dominates

the processing and the number of PE's exactly matches the number of elements in the pyramid data structure.

REFERENCES

1. K. E. Batcher, "Design of a Massively Parallel Processor," *IEEE Transactions on Computers* C-29(9) pp. 836-840 (September 1981).
2. A. P. Reeves, "On Efficient Global Information Extraction Methods For Parallel Processors," *Computer Graphics and Image Processing* **14** pp. 159-169 (1980).
3. A. P. Reeves, "Parallel Pascal: An Extended Pascal for Parallel Computers," *Journal of Parallel and Distributed Computing* **1** pp. 64-80 (1984).
4. A. P. Reeves, "The Anatomy of VLSI Binary Array Processors," in *Languages and Architectures for Image Processing*, ed. M. J. B. Duff and S. Levialdi, Academic Press (1981).
5. R. W. Gostick, "Software and Algorithms for the Distributed-Array Processor," *ICL Technical Journal*, pp. 116-135 (May 1979).
6. NCR Corporation, *Geometric Arithmetic Parallel Processor*, NCR, Dayton, Ohio (1984).
7. A. P. Reeves and C. H. Moura, "Data Manipulations on the Massively Parallel Processor," *Proceedings of the Nineteenth Hawaii International Conference on System Sciences*, pp. 222-229 (January, 1986).
8. P. J. Burt, "Fast Filter Transforms for Image Processing," *Computer Graphics and Image Processing* **16** pp. 20-51 (1981).

PYRAMIDAL TRANSFORMS IN IMAGE PROCESSING AND COMPUTER VISION

Ph. W. Besslich

Electrical Engineering Section, University of Bremen
D- 2800 Bremen 33, F. R. of Germany

ABSTRACT: This chapter deals with a class of transform algorithms suitable for the generation and processing of pyramidal data. The transformations make use of radix-2^k signal-flow graphs and "in-place" processing. The scheme is based on a hierarchical ordering of a $2^n \times 2^n$ data array in memory for radix-2 signal-flow graph processing or, alternatively, on linewise stored data using incomplete radix-2^k graphs ($k=1,2,\ldots,n$). The generation of pyramidal data structures is a special case of a more general class of 2-D transformations that calculate 2-D transform coefficients hierarchically, i.e. from the coefficients of subareas. Various useful global or local transformations may be implemented under the hierarchical scheme. Pyramidal data structures are shown to be a special case of hierarchical transforms. For instance, each node level of an averaging pyramid corresponds to a hierarchy level of a reversible transformation. From the class of orthogonal transformations the 2-D Walsh-Hadamard transform (WHT) complies with the stipulations of hierarchical generation of coefficients. Local versions of pyramidal radix-2^k transformations provide windows of size $2^k \times 2^k$. Windows of odd-numbered dimensions require radix-p based transformations (p=prime number > 2). Local pyramids using weighted and overlapping window operations are shown to allow various weighting functions of the window, e.g. a Gaussian weighting. Pyramidal transform algorithms may be supported by special hardware. An architecture is proposed that uses spatially parallel processing for each pyramid layer and macro pipelining among the layers.

NATO ASI Series, Vol. F 25
Pyramidal Systems for Computer Vision
Edited by V. Cantoni and S. Levialdi
© Springer-Verlag Berlin Heidelberg 1986

CONTENTS

1. Introduction

2. Computational Procedures

   2.1 Signal-flow graph operations
   2.2 Similarity transformations based on the WHT
   2.3 Signal-flow graphs for linewise stored data

3. Pyramidal Transformations

   3.1 Generation of intensity averaged pyramids
   3.2 Local transforms using radix-2^k signal-flow graphs
   3.3 Signal-flow graphs for odd-numbered window sizes
   3.4 Construction of weighted and overlapping pyramids

4. Global Hierarchical Transformations

   4.1 Hierarchical generation of coefficients
   4.2 Examples of global hierarchical transformations

5. Parallel Architecture for Pyramidal Transformations

   References

1. INTRODUCTION

Most problems in image processing and pattern recognition require the processing of an immense amount of data in relatively short time. An example of such a task is real-time processing for computer vision. In order to cope with these severe requirements, various approaches to parallel and multi-resolution processing of iconic data have been suggested [1-4] . A promising concept to solve this problem is multi-resolution representation of data in the form of a pyramid [3-6]. Great efforts have been undertaken to develop powerful algorithms that allow the desired information to be extracted efficiently from this data structure [7-11] . An equally important issue is the development of suitable architectures and hardware on which the algorithms can be executed. Different solutions have been proposed and several of them have been realized [12-18] . Last but by no means least, appropriate software must be provided if such a computing structure is to be put to work economically. A considerable amount of work has been devoted to this subject [19-21] .

Pyramids are data structures. In order to put them to an efficient use we have to connect processors to the pyramidal memory structure. There are different ways to do so:

- To build a pyramid of processors,
- To use flat arrays of processors,
- To employ a single or several processors per layer,
- To employ only one processor for the pyramid.

The first two of these possibilities are discussed in other papers contained in this volume, while the last of the four possibilities is essentially the simulation of a pyramid machine on a general purpose computer. In this chapter we are concerned with connecting one or several processors to each pyramid layer. The processors are arranged so that the generation and processing of pyramidal data is optimized with respect to throughput.

It is the author's firm belief that the advantages of pyramid machines can be exploited only if algorithms and architectures are simultaneously taylored for this purpose. Hence, in this contribution we consider the possibility of using hierarchical transformations for both generating and processing the pyramidal data as well as for implementation of various algorithms to be run on a pyramidal system. The transform operations that are involved in this method are of the radix-2 "in-place" type. This kind of processing is usually employed, for instance, for the execution of the fast Fourier transformation (FFT). In these

procedures 2 data are pulled from memory and combined. Then the results are stored back into the same memory locations, since the original data are no longer needed. Though the significance of in-place operations has dropped in view of falling memory prices, we show how the allotment of memory locations in radix-2 algorithms can be exploited for efficient parallel processing. Another motivation for using this type of algorithms is reduction of computational complexity from $O(N^2)$ to $O(N \cdot \log_2 N)$. Moreover, radix-2 algorithms will be shown to ideally support generation and processing of pyramidal data.

In particular we consider a certain class of 2-D transformations in which coefficients are obtained hierarchically, i.e. from those of subareas. Hierarchical processing of 2-D data may be described in terms of a pyramid of data in which each level represents a different resolution. Working with a pyramid-like data structure suggests the use of cellular processors. A few such cellular pyramid machines have been designed and constructed [16-18] . Another approach uses a smaller number of microprocessors or signal processors rather than one cellular processor for each picture element. Some very general or reconfigurable architectures have also been realized [13-15] . In recent papers pyramid architectures that support radix-2 operations have been reported [22, 23] . This contribution is mainly devoted to the construction and processing of pyramidal data using a signal-flow graph approach. Though the radix of the signal-flow graph may be chosen arbitrarily, we focus our attention on a radix of 2 (or integral powers of 2) because this choice fits the usual dimensions of pictures in terms of picture elements. Hence, we are concerned with 2-D radix-2^k transformations ($k \in \mathbb{N}$) that allow hierarchical generation of coefficients. The aim is therefore to process the tree-like structure of Fig. 1 by radix-2 algorithms so that the original memory space is used for side information that may serve for a reversed (top-down) operation. Since the pyramidal data themselves, as well as

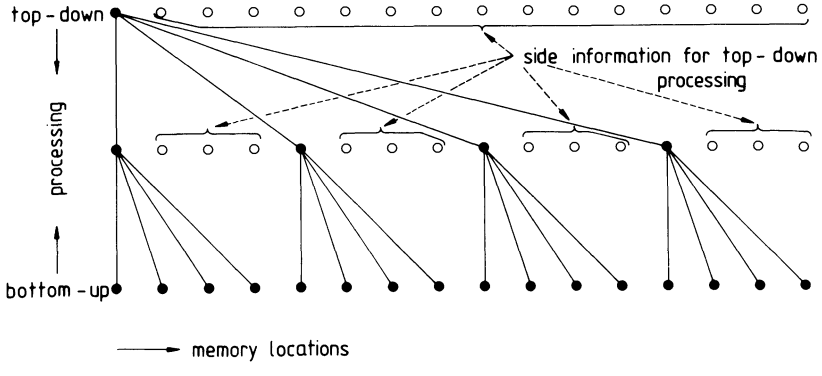


Fig. 1 . Example of hierarchical processing: quadrant pyramid.

their side information (cf. Fig. 1) are coefficients of a pyramidal transformation, we may merge this procedure with other transformations to obtain low-complexity operators. Further, two types of signal-flow graphs for executing the transformations are discussed: for hierarchical and for linewise storage of the pictorial data. Applications of the scheme are in the field of linear and non-linear operations for image processing and computer vision.

2. COMPUTATIONAL PROCEDURES

2.1 Signal-flow Graph Operations

Before dealing with transform domain operations and their similarities to the object domain, we consider methods for ordering data hierarchically in memory and for exploiting their organization in the 2-D processing. Hierarchical memory organizations of pictorial (2-D) data have been suggested by several authors [24,25]. It has been pointed out previously that hierarchically indexed $2^n \times 2^n$ arrays are suitable structures for iterative radix-2 signal-flow operations [26]. The most important property of this index system is that data elements are located in

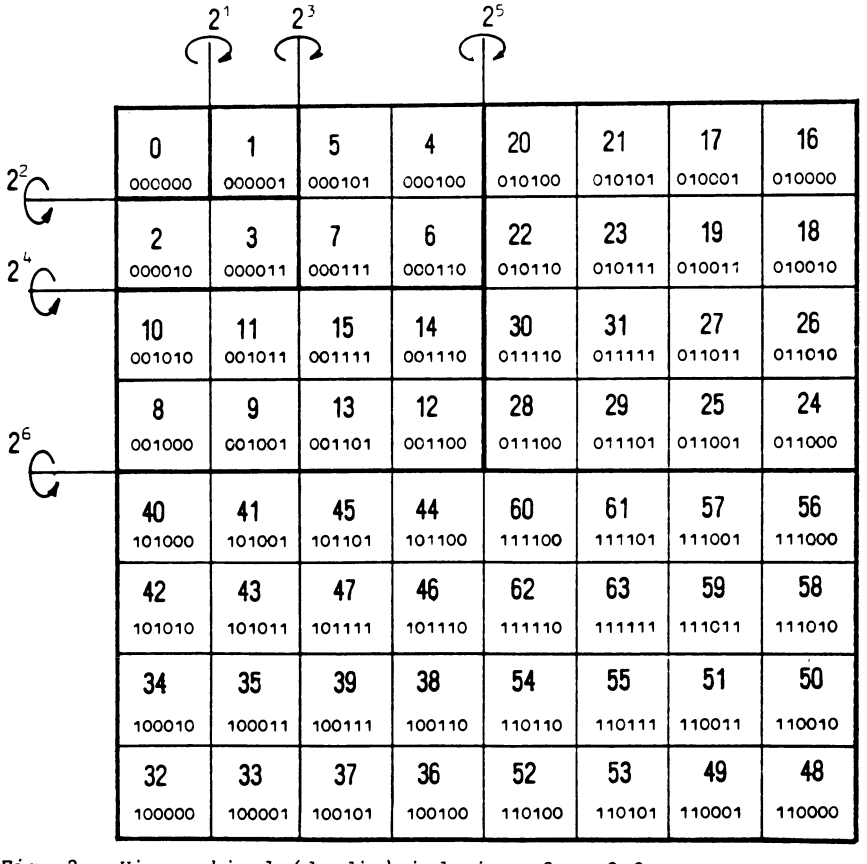


Fig. 2 . Hierarchical (dyadic) indexing of an 8x8 array.

memory in an order of hierarchically ordered subsquares. Data elements having index (relative address) Nos. 0-3, 4-7, 8-11, etc. form all the 2x2 subsquares, Nos. 0-15, 16-31, 32-47, etc. form all the 4x4 subsquares, and so on. A method for the generation and processing of dyadically ordered 2-D data has been proposed recently [26]. In this contribution we describe pyramidal transformations, i.e. the hierarchical generation of coefficients from those of lower order. First we review briefly signal-flow graph operations using the dyadic index (address) system.

Let 2^{2n} data elements be indexed by the 2n-digit binary numbers 00...0 to 11...1. Further, let the data be arranged in an array as shown in Fig. 2. This pattern of indices is obtained by successively unfolding a square (or double square) alternatively along its right hand side and its lower edge. The already existing indices are mirrored and for the new subarea a leading (binary) 1 is added to the index. Note that the indices of those cells located symmetrically to a folding axis differ in only one bit, i.e. they have Hamming distance 1. Or, in other words, the pattern of 2^{2n} binary coded cells is a mapping of a hypercube of dimension 2n. Each iteration of the associated signal-flow graph is to combine elements having Hamming distance 1, i.e. symmetrically located elements, by a "butterfly" operation. For a simple 2x2 array the corresponding signal-flow graph is shown in Fig. 3. The horizontal lines in these graphs correspond to memory locations. In $\log_2 m$ iterations their contents $(x_i; i=0,1,\ldots,2^{m-1})$ is transformed into the coefficients y_i without using additional memory. Hence, the name "in-place" transformation. Fig. 3 contains 4 basic butterflies, each one composed of the arrows a,b,c,d where a through d are multipliers. We consider them to be elements of a matrix M_1:

$$\mathbf{M}_1 = \begin{bmatrix} a & c \\ b & d \end{bmatrix} \qquad (1)$$

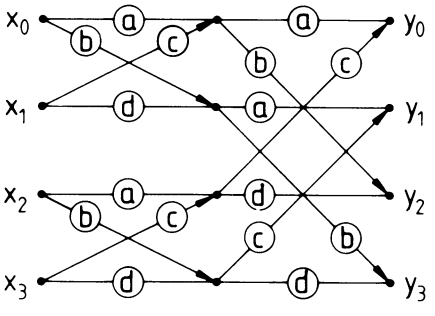


Fig. 3. Radix-2 signal-flow graph of a 2x2 array.

We can now construct higher order matrices corresponding to higher order signal-flow graphs using the Kronecker matrix product

$$\mathbf{M}_q = \underbrace{\mathbf{M}_1 \otimes \mathbf{M}_1 \otimes \ldots \otimes \mathbf{M}_1}_{q} = \left\{\mathbf{M}_1\right\}^q , \qquad q \in \mathbb{N} \tag{2a}$$

where

$$\mathbf{M}_2 = \mathbf{M}_1 \otimes \mathbf{M}_1 = \begin{bmatrix} a\begin{bmatrix} a & c \\ b & d \end{bmatrix} & c\begin{bmatrix} a & c \\ b & d \end{bmatrix} \\ b\begin{bmatrix} a & c \\ b & d \end{bmatrix} & d\begin{bmatrix} a & c \\ b & d \end{bmatrix} \end{bmatrix} \tag{2b}$$

Using (1) and (2) we can formulate the transformation

$$\mathbf{y} = \mathbf{M}_q \cdot \mathbf{x} , \qquad q \text{ even}, \tag{3}$$

where $\mathbf{x}$ and $\mathbf{y}$ are the original and the processed data vector, respectively. Hence we use the well-known (1-D) radix-2 signal-flow graph (Fig. 3) to process 2-D data ordered dyadically in a vector. Note that the signal-flow graph is composed of q iterations, q/2 of which serve for horizontal and q/2 for vertical transformations. Hence, separability of the 2-D transformation is a necessary condition.

If $\mathbf{M}_1$ is an invertible square matrix, then we have:

$$\mathbf{M}_q^{-1} = \left\{\mathbf{M}^{-1}\right\}^q , \qquad q \in \mathbb{N} \tag{4}$$

Consequently, we may invert the operation provided $ad \neq bc$.

2.2 Similarity Transformations Based on the WHT

Let a linear transformation be described by

$$\mathbf{M} = \mathbf{T}^{-1} \cdot \mathbf{N} \cdot \mathbf{T}. \tag{5}$$

Then the matrices $\mathbf{M}$ and $\mathbf{N}$ are called similar. If $\mathbf{T}$ is an orthogonal transformation, $\mathbf{M}$ and $\mathbf{N}$ are said to be orthogonally similar. A special case arises if $\mathbf{M}$ is real and symmetric, because $\mathbf{N}$ can then be shown to be diagonal, the elements being the eigenvalues of $\mathbf{M}$. In our context, we use the Walsh-Hadamard transform (WHT) matrix $\mathbf{T} = \mathbf{H}$. Note that the WHT itself is a special case of (1), where a=b=c=1 and d= -1.

The purpose of these similarity considerations is to emphasize the possibility of rewardingly executing the operations in the WHT domain. Hence, we consider similar radix-2 operations on the WHT coefficients, i.e. transform domain operations which correspond to radix-2 operations applied to the data itself. To facilitate this, use is made of the fact that it suffices to consider the case q=1, i.e. the 2-element case (2). Fig. 4 illustrates the situation: Embedded between a forward and an inverse 2-element WHT the radix-2 operation (α , β , γ , δ) is to produce the same result as is obtained from a (a, b, c, d) radix-2 operation directly.

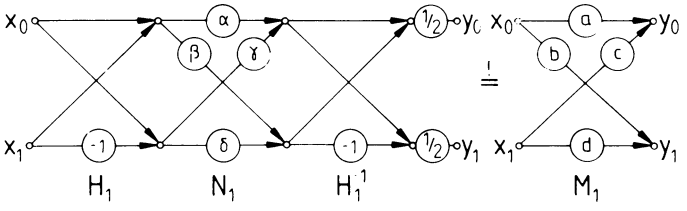


Fig. 4 . Similarity of 2-element graphs under the WHT.

For the relationship between (α , β , γ , δ) and (a,b,c,d) we have [27] :

$$\alpha = (a+b+c+d)/2 \qquad \beta = (a-b+c-d)/2$$

$$\gamma = (a+b-c-d)/2 \qquad \delta = (a-b-c+d)/2 \tag{6}$$

Rewriting (5) we obtain

$$\mathbf{M} = \mathbf{H}^{-1} \cdot \mathbf{N} \cdot \mathbf{H} \ , \tag{7}$$

where **N** is a Kronecker power of

$$\mathbf{N}_1 = \begin{bmatrix} \alpha & \gamma \\ \beta & \delta \end{bmatrix} . \tag{8}$$

Equation (7) exhibits an asset of this in-place method: Each problem may be processed either in the data domain or in the spectral domain, depending on convenience or complexity considerations. Speaking in terms of system theory we have the choice of multiplying the spectral representation of the data by a spectral response, or alternatively carrying out a convolution in the data domain. From the point of view of computational complexity the latter becomes attractive if it can be computed with $0(2^q)$ operations. We will show that this is in fact possible.

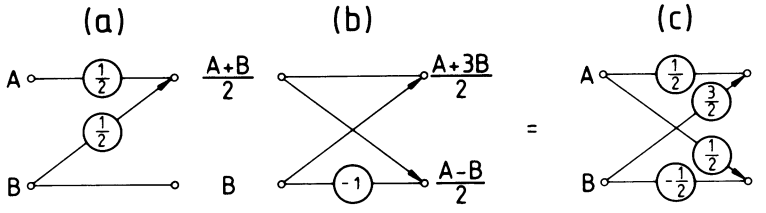


Fig. 5 . Example of combined operations in the object domain: (a) Pyramid generation, (b) WHT, (c) WHT of pyramid.

An example of a combined transformation, both in the object and in the transform domain, is illustrated in Fig. 5 and Fig. 6. Fig. 5(a) depicts the basic flow graph $\mathbf{P}_1$ for pyramid generation, while Fig. 5(b) is the basic WHT flow graph $\mathbf{H}_1$. Their combination is shown in Fig. 5(c). Computation of 2i iterations generates the 2-D WHT coefficients of the i-th pyramid level. In terms of matrix representation we have:

$$\mathbf{H}_1 \cdot \mathbf{P}_1 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \cdot \begin{bmatrix} 1/2 & 1/2 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1/2 & 3/2 \\ 1/2 & -1/2 \end{bmatrix} \tag{9}$$

The similar operations in the WHT domain are shown in Fig. 6 (a) through (c), where the superscript s denotes transform domain:

$$\mathbf{H}_1^s \cdot \mathbf{P}_1^s = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \cdot \begin{bmatrix} 1 & -1/2 \\ 0 & 1/2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1 & -1 \end{bmatrix} \tag{10}$$

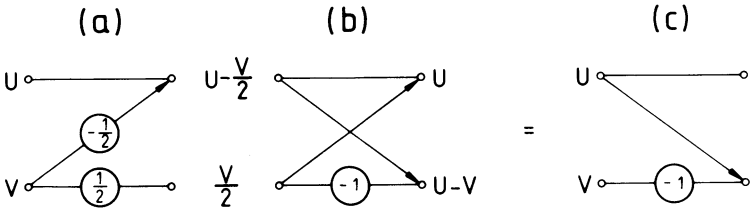


Fig. 6 . Similar operations to those depicted by Fig. 5.

2.3 Signal-flow Graphs for Linewise Stored Data

Section 2.1 was based on the assumption of dyadically ordered data in memory. This order is normally not present in stored image arrays. Images are usually

stored in a line-scanned sequence. Hence, radix-2 generation of transform coefficients is not directly applicable. One possibility of remedying the situation is to relocate data in memory. In-place algorithms that change memory mapping from line order to dyadic order and vice versa may be devised [28] . However, for many applications data rearrangement or address calculations are not a satisfying solution, particularly if considered from the point of view of processing time. Therefore we develop a modified signal-flow graph which allows direct execution of the algorithms on data mapped to memory in line-scanned form [28] .

The aim is to find new signal-flow graphs which produce the same 2-D coefficients for line-sequential data as the radix-2 signal-flow graph does for dyadically ordered data, except for a possible change in their order. A simple and pragmatic way to achieve such a mapping is first to rearrange the initial data accordingly, and then to redraw the signal-flow graph, successively for each iterative step. Fig. 7 shows an example for a 4x4 array. The 1st and the 2nd iterative step are of radix-2. But the 3rd and 4th iteration use incomplete radix-4 signal-flow graphs. Each radix-4 iteration requires as many two-by-two combinations as the corresponding radix-2 iteration does. Hence, the complexity remains the same. Because the symmetries are different from those of the dyadic index system, certain butterflies are "upside down", i.e. the multipliers a and d are interchanged, and so are b and c. The result obtained for the 4x4 data array can be generalized

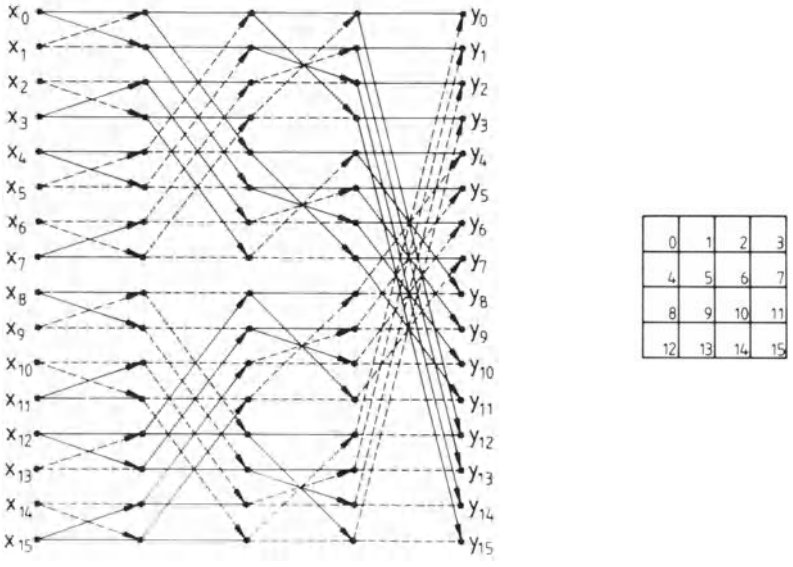


Fig. 7 . Signal-flow graph of a linewise addressed 4x4 array (solid lines: a and b, dotted lines: c and d).

in a straightforward manner. A $2^n \times 2^n$ image requires 2n iterative steps: two each of radix $2^1, 2^2, \ldots, 2^n$.

Note that the transformation is separable and invertible. In order to reverse a linear operation, or to devise a topdown process, the iterative steps must be taken in reversed order with the multipliers replaced by the inverse ones, unless the matrix $\mathbf{M}_1$ is singular. The resulting signal-flow graph is of identical geometrical shape, i.e. for forward and reverse transformation the very same 2^{2n-1} multipliers are different from zero. The possibility of directly computing hierarchical coefficients (e.g. pyramidal data) from a line-scanned sequence is of major importance for practical applications of pyramidal transforms.

3. PYRAMIDAL TRANSFORMATIONS

3.1 Generation of Intensity Averaged Pyramids

Pyramids are data structures constructed in order to efficiently execute multiresolution algorithms. In this contribution we consider quadrant based pyramids only. Moreover, pyramids may either be non-overlapping or overlapping. Further, the information stored on the higher layers may be obtained by simple averaging or by other operations. Hence, in their simplest form pyramids are quite similar to a quadtree structure, but provide the full information of the various levels of the tree simultaneously. We confine ourselves to 2-D data arrays of size $2^n \times 2^n$, where $n \in \mathbb{N}$. Such arrays of data are appropriate for processing by radix-2^k signal-flow graphs ($1 \leq k \leq n$). Note that other radices (e.g. radix-3) may be useful alternatives. The simplest case of a $2^n \times 2^n$ pyramid is one in which each level of next higher order is a non-overlapping reduced-resolution version of its lower neighbor. It is constructed by averaging a square of 4 adjacent elements to form one new element. This kind of pyramid corresponds to a quadtree, i.e. a tree in which each node has 4 children. Nodes that have children of uniform color are terminating leaves of the tree. Hence, in contrast to the pyramidal representation, a tree does not necessarily grow branches through all the levels. The relationship between an averaging square pyramid and the corresponding quadtree is illustrated by an example depicted in Fig. 8 (a) and (b).

We now apply the radix-2 signal-flow graph method (cf. section 2) to the construction of averaging quadrant pyramids. Let $a = c = 1/2$, $b = 0$ and $d = 1$.

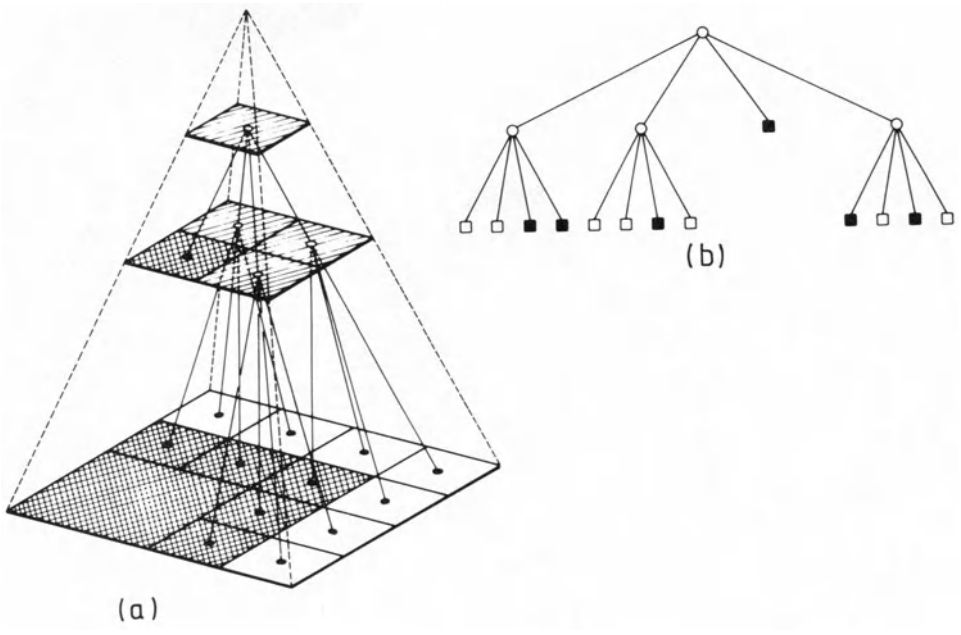


Fig. 8 . Example of a simple pyramid (a) and its associated quadtree (b).

The basic signal-flow graph of the pyramid generating transformation and its inverse are depicted by Fig. 9. Applying it to a 2x2 array we obtain the following coefficients (cf. Fig. 3):

$$
\begin{aligned}
y_0 &= \frac{x_0 + x_1 + x_2 + x_3}{4} \\
y_1 &= \frac{x_1 + x_3}{2} \\
y_2 &= \frac{x_2 + x_3}{2} \\
y_3 &= x_3
\end{aligned}
\qquad (11)
$$

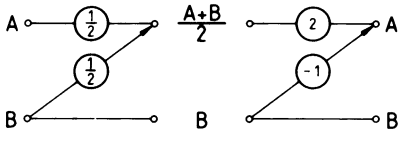


Fig. 9 . Basic pyramid generating graph and its inverse.

The first of these coefficients (y_0) averages the four elements x_0 to x_3 while the three others serve for reconstruction of the leaves x_0 to x_3 from the average y_0. Any two iterations of this basic flow graph generate a level of a quadtree (or pyramid), while three iterative steps are required for an octree. Alternatively, quadtrees and octrees may also be processed by radix-4 and radix-8 algorithms, respectively. For convenience we restrict the discussion to multiple radix-2 operations. An example of a bottom-up and a top-down pyramid generating transformation of a 4x4 image with data elements A through P are shown in Fig. 10 and Fig. 11, respectively.

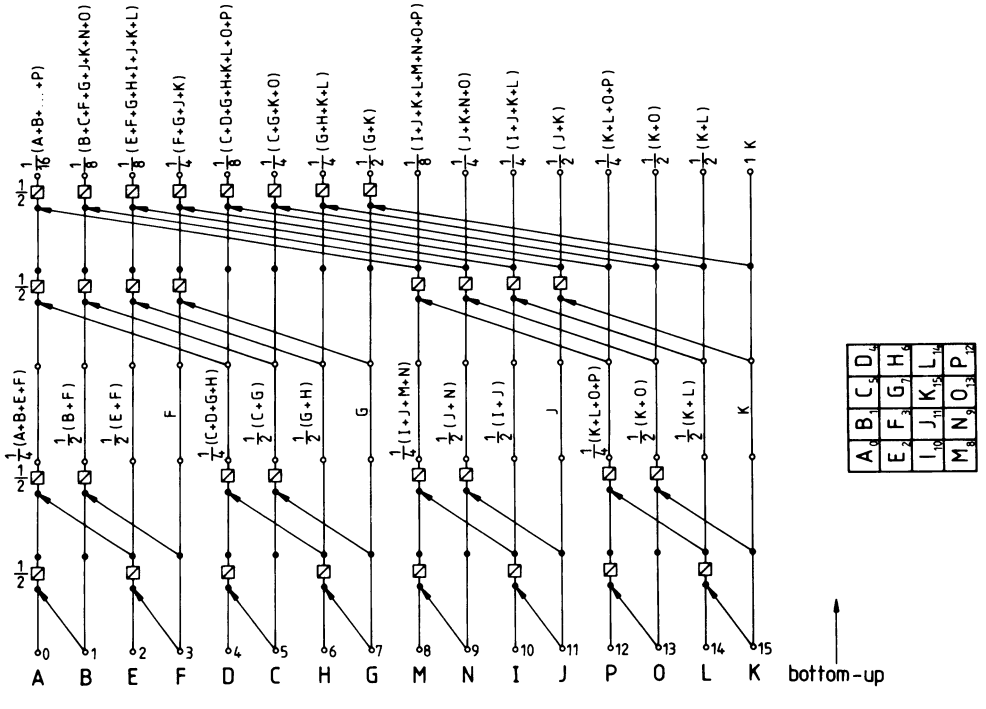


Fig. 10. Signal-flow graph for bottom-up processing of averaging pyramids.

Let the x_j represent the "color" of a pixel, (e.g. for the binary case: white:= 0 and black: = 1). The color of nodes of the i-th node level is obtained after 2i iterations, represented by their normalized averages and given by any 4i-th coefficient. The other coefficients are of no importance for the bottom-up procedure. They are, however, necessary to preserve all input information which is needed for a top-down process.

Let the image consist of NxN = 2^nx 2^n pixels stored in 2^{2n} random access memory locations. The procedure described here generates an invertible and easy

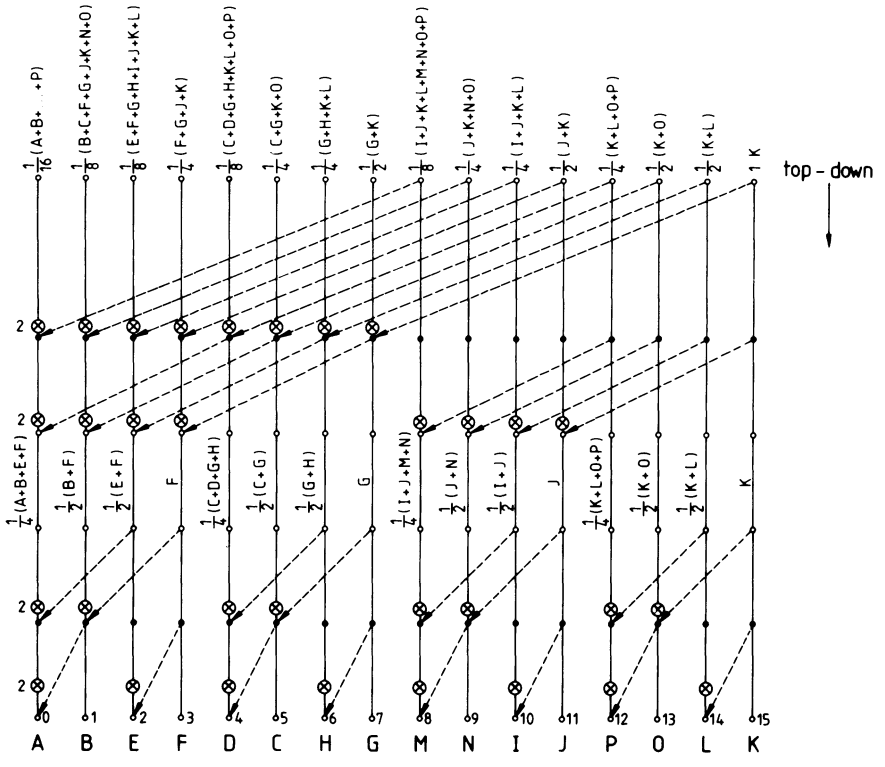


Fig. 11. Signal-flow graph for top-down processing of averaging pyramids.

to read pyramid representation in the very same memory locations. Fig. 1 illustrates a pyramidal transformation. If the image is given on the pixel level, a bottom-up procedure is required to get the apex of the pyramid (the trunk of the quadtree). Fig. 9 depicts the basic signal-flow graph and its inverse. The output y_0 accounts for the average of all inputs, i.e. the apex or the trunk. The other coefficients are needed to allow an inverse processing in order to develop the lower levels of the pyramid or the branches and leaves of the tree. Any two forward or inverse iterative steps generate the elements of the next higher or lower level, respectively. Since the operation is linear and non-singular one can easily keep changing levels. Hence the algorithm may be considered to produce "virtual" pyramids or quadtrees because only one level is present at a time. From Fig. 11 it can be observed that during the various iterative steps of the top-down processing memory cells contain averages of subsquares and hence account for the information of the various pyramid levels. In each basic butterfly only 1 addition and 1 shift is involved. Hence, the bottom-up operation requires $2^{2n} \cdot \log_2(2^{2n}) = 2N^2 \cdot \log_2 N$ operations.

To generate a pyramid from linewise indexed (addressed) data we use the general scheme depicted by Fig. 7. Application of the basic signal-flow graph of the pyramid generating transformation (Fig. 9) to the graph of Fig. 7 yields a new processing scheme for linewise stored images. Fig. 12 shows an example of a 4x4 array.

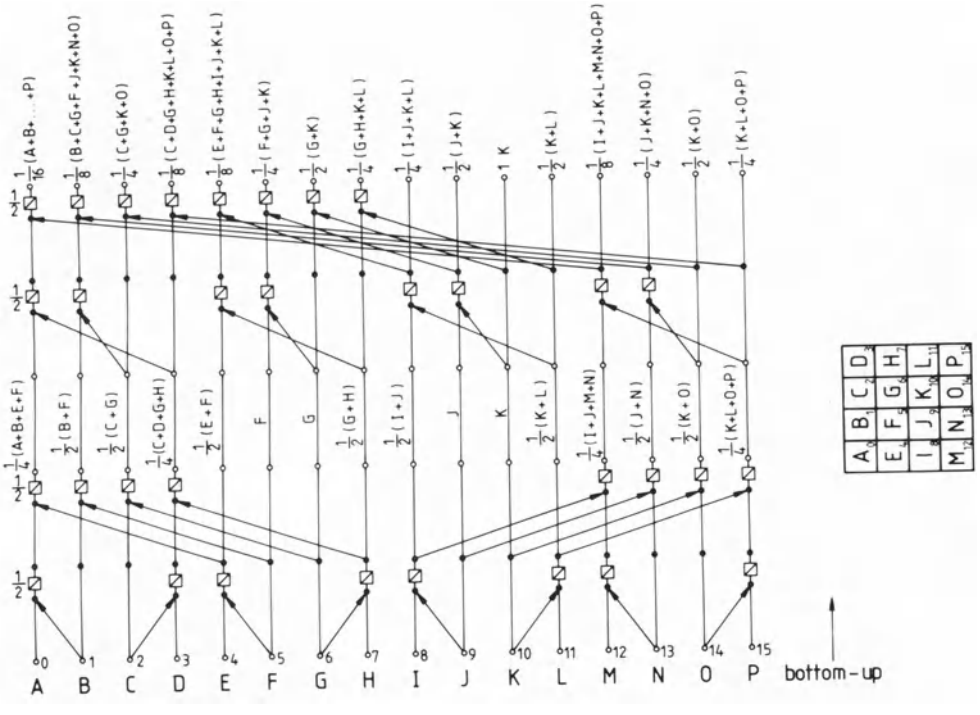


Fig. 12. Pyramid generation from linewise addressed data.

3.2 Local Transforms Using Radix-2^k Signal-flow Graphs

We consider now local transformations in which each data element (pixel) interacts only with those others which are located within a local 2-D window. In other words: each pixel is provided with its individual window. To give a better idea of how these local signal-flow graph algorithms work, we consider an introductory example of a 4x4 window opened for pixel No. 28 in a linewise indexed 8x8 picture matrix as illustrated by Fig. 13. The associated signal-flow graph for linewise addressed data is depicted in Fig. 14. Note that the signal-flow graph provides an appropriate 2-D window for each data element. It is seen that the graph consists of two different matrices $\mathbf{M}_1$ and $\mathbf{M}_0$, where for $\mathbf{M}_0$ we have a=d=1 and b=c=0. The operation is invertible, provided $\mathbf{M}_1$ is not singular. Due to the symmetries involved in the dyadic address system the multipliers a

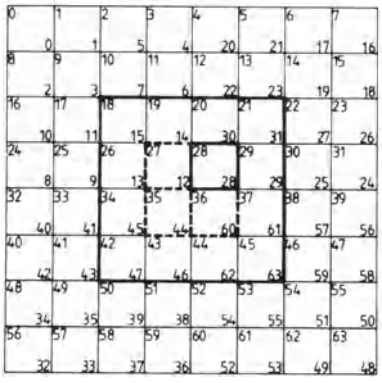


Fig. 13. 4x4 window of element 28 (also of 27, 35, and 36). Top left figures: linewise addressing, bottom right: hierarchical addressing.

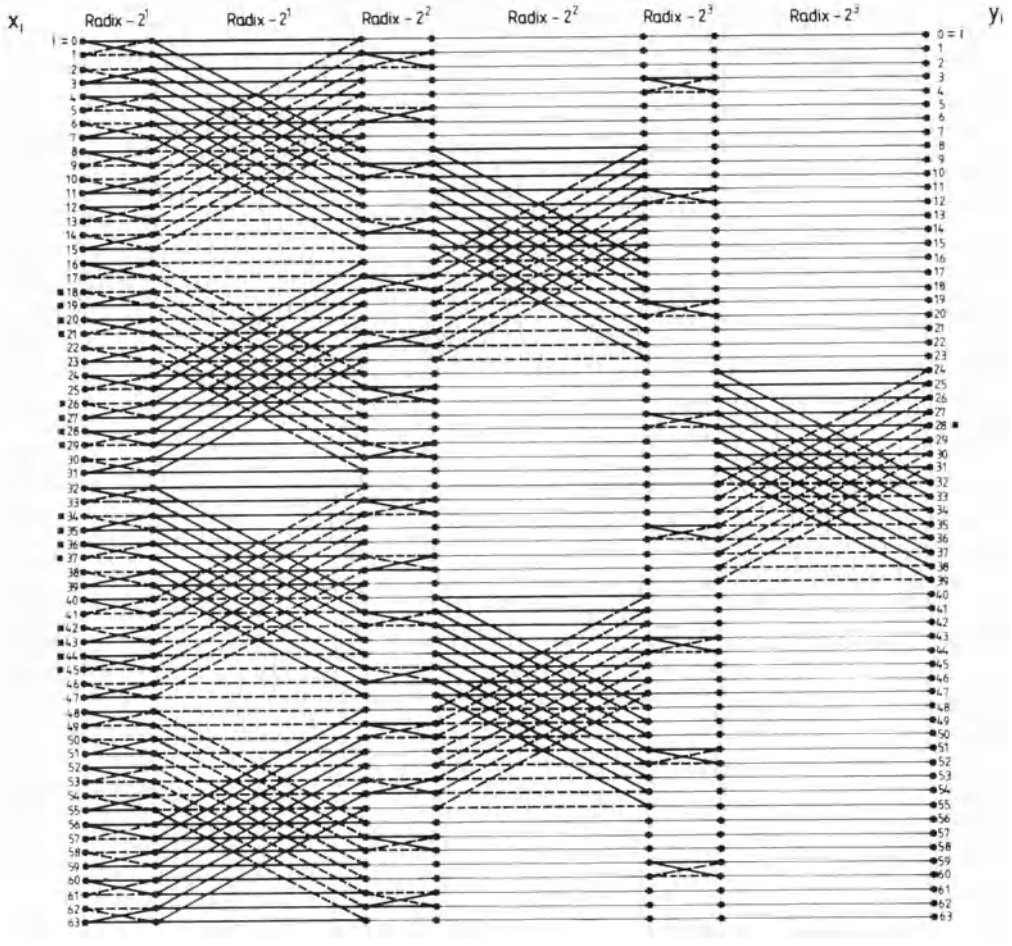


Fig. 14. Local signal-flow graph (linewise addressed) of a 8x8 image. Data x_j contributing to coefficient y_{28} are indicated.

and d as well as b and c are interchanged in some of the butterflies (cf. section 2.3). In order to indicate their orientation, a and d are drawn with dotted lines. The signal-flow structure described here generates (non-symmetrical) 4x4 windows. The four pixels in each 2x2 subsquare have the same window (cf. Fig. 13). More details (e.g. how to generate 3x3, 5x5, etc. windows) will be discussed below.

Let us first consider the properties of the 4x4 window transformation. The whole process of the in-place transformation may be thought of as a matrix multiplication. The two factors are a vector that contains the 2-D data in some lexicographical (dyadic or linewise) order and a transform matrix [28] . Decomposition into the various iterative steps can be obtained if the transform matrix can be factored suitably. An example is the Kronecker product matrix M_q, cf. (2) and (3). In the case of a window operation the transform matrix takes the form of a sparse matrix, the non-zero elements of which are centered near the main diagonal. Speaking in algebraic terms, the local transformation may be described as a multiplication of a row vector x_r of image data with a transform matrix L_i

$$y_{ri} = x_r \cdot L_i \ , \tag{12}$$

or alternatively by the product of L_i and a column vector x_c of the image data

$$y_{ci} = L_i \cdot x_c \tag{13}$$

The matrix L_i may be split into a product of q factors representing the q iterations:

$$L_i = L_{i1} \cdot L_{i2} \cdot \ \dots \ \cdot L_{iq} \tag{14}$$

Since some of the submatrix products in (14) turn out to be commutative, only two different products L_1 and L_2 are possible. Permutation of submatrices is equivalent to reordering the q different iterative steps. Each of the two local transforms L_1 and L_2 can be implemented by q!/2 permutations of iterations in the signal-flow graph. Hence, we have four configurations leading to four different windows: L_1 and L_2 multiplied from the left by the row vector x_r or else multiplied from right by the column vector x_c. The four different windows obtained are illustrated by Fig. 13. Each of the four in-place transformations is invertible. For this purpose the iterative process is to be performed in reverse order using the inverse of the elements a through d of the basic butterflies [28] .

An interesting and useful feature of these incomplete radix-2^k signal-flow graphs is the possibility of reducing the number of iterations to four, independently of the size of the picture. From Fig. 14 it can be observed that the 3rd and the 5th iteration can be combined without any mutual interference. Likewise we may combine iterations No. 4 and No. 6. The reduction to 4 iterations does not, however, affect the total number of operations required.

Consider a $2^n \times 2^n$ image to be processed by a 4x4 local operator (cf. Fig. 14). The first two iterative steps consist exclusively of **M** butterflies, each accounting for 2^{2n} operations (=1 multiplication + 1 addition). The next two iterative steps employ only half the number of **M** butterflies, the rest is of $\mathbf{M}_0$ type. The latter do not rquire any operation. In this way the number of operations keeps reducing by a factor 2 for any two iterations. Hence, for the 4x4 local transformation.

$$O_L = 2 \cdot 2^{2n}(1+1/2+1/4+ \ldots +1/2^{n-1}) < 2^{2n+2} = 4N^2 \qquad (15)$$

operations are required to

$$O_G = 2^{2n} \cdot 2n = 2N^2 \cdot \log_2 N \qquad (16)$$

for the global transformation, i.e. reduction by a factor n/2. For example: An image of size 512x512 picture elements requires 18 iterations with 4 718 592 operations, while the local 4x4 transformation has only 4 iterations and 1M operations.

3.3 Signal-flow Graphs for Odd-numbered Window Sizes

For certain applications the natural unsymmetry of 4x4 windows is not appropriate. Hence, we also aim for windows that are symmetric with respect to a central element, i.e. windows of size 3x3, 5x5, etc.

Local transforms developped in the previous sections use incomplete radix-2^k graphs. Combinations of elements that are neighbors in a row or in adjacent rows require k = 1. Iterations that are to combine elements of more distant rows require higher values of $k \leq 2^n$. In the case of 4x4 local operations we "slimmed down" the radix-2^k signal-flow graph so that it could be condensed to only 4 iterations. In this approach signal-flow graphs are derived from radix-2^k. Hence, the windows cannot have a central element. To obtain windows of odd-numbered size the graph must be of odd radix.

We consider now 3x3 local operations. First we observe that coefficients are obtained by combining three elements horizontally and vertically. Hence, two iterations of radix-3 are sufficient: One iteration serves for 3x3 combining in horizontal direction, the other one for the vertical direction. In other words: combinations of the even and the odd-numbered iterations (cf. Fig. 14) form a radix-3 based signal-flow graph that generates 3x3 windows centered for each element. The general structure of these 2-iteration signal-flow graphs is illustrated by Fig. 15(a). It is evident from the figure that weights assigned to the

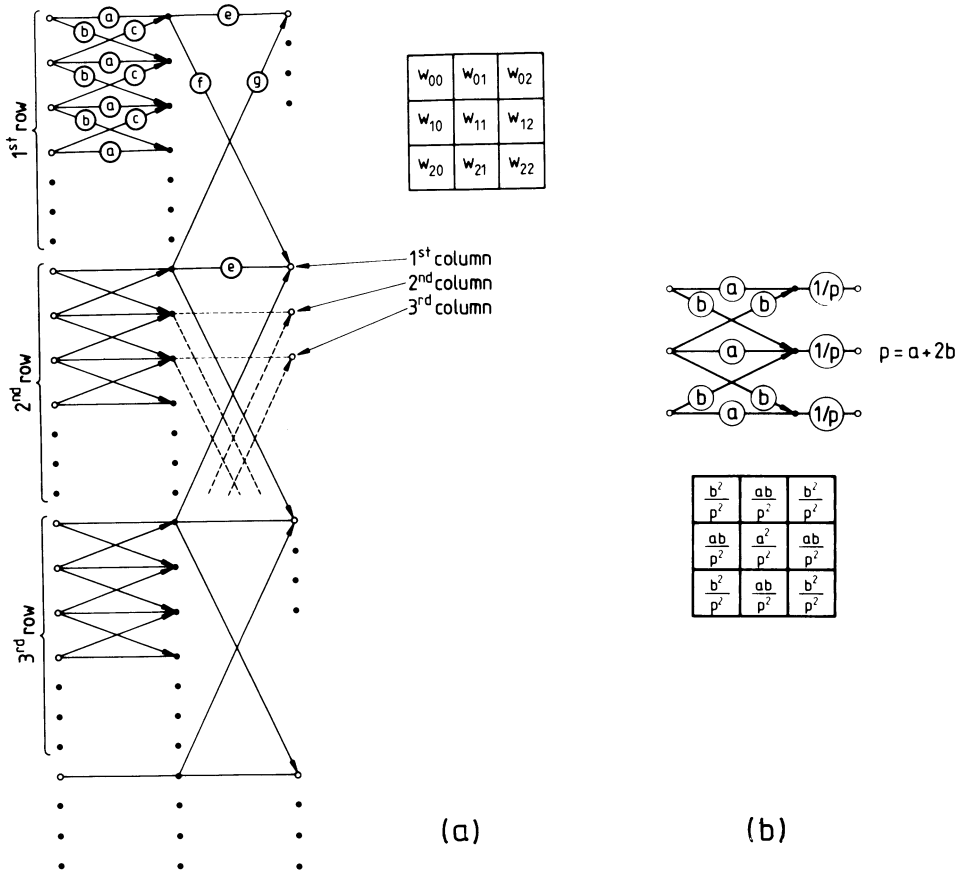


Fig. 15(a) . Signal-flow graph and weights for local 3x3 transform.
(b) . Symmetric 3x3 weighting.

nine elements of the window cannot be chosen independently. They are restricted by the following conditions, where the weights w_{00} through w_{22} and the multipliers a through f correspond to Fig. 15(a).

$$\frac{a}{b} = \frac{w_{01}}{w_{00}} = \frac{w_{11}}{w_{10}} = \frac{w_{21}}{w_{20}} \qquad \frac{e}{f} = \frac{w_{10}}{w_{00}} = \frac{w_{11}}{w_{01}} = \frac{w_{12}}{w_{02}}$$

$$\frac{a}{c} = \frac{w_{01}}{w_{02}} = \frac{w_{11}}{w_{12}} = \frac{w_{21}}{w_{22}} \qquad \frac{e}{g} = \frac{w_{10}}{w_{20}} = \frac{w_{11}}{w_{21}} = \frac{w_{12}}{w_{22}} \tag{17}$$

The linear dependencies imposed by (17) are a major restriction on choosing the weights. However, it turns out that most of the smoothing and edge detecting operators can be implemented under the constraints (17).

3.4 Construction of Weighted and Overlapping Pyramids

The simple averaging pyramid of section 3.1 exhibits sharp cutoffs of the square regions which may not be desirable in some applications. Therefore weighted and/or overlapping pyramids have been used by several researchers [29,30]. Burt has used a "Gaussian" pyramid in which the spatial weighting function approaches a Gaussian shape as the reduction progresses. Hence, the reduced versions of a 2-D signal are (Gaussian) low-pass filtered signals. The differences of two (equal sized) Gaussian versions of subsequent resolution levels are band-pass filtered or "Laplacian" signals. In this section we show how the local version of the signal-flow graph scheme can be used to compute overlapping and weighted pyramids.

In order to obtain overlapping and weighted pyramid data, the distance measure of which is equal in horizontal and in vertical direction, we use symmetric radix-3 signal flow graphs as depicted in Fig. 15(b). The factor 1/(a+2b) serves for normalization. Weights assigned to any 3x3 neighborhood are as illustrated by Fig.15(b). If the signal-flow graph is of the structure shown by Fig. 15(a), this procedure generates weighted linear combinations from data within 3x3 windows centered at each element. To construct an overlapping pyramid only every other picture element in rows and columns is used for the first pyramid level above bottom. Hence, the signal-flow graph needs only 1/4 of the original memory locations of level i=1 (bottom). In this way the reduction process of the weighted and overlapping pyramid keeps progressing. The overlapping and weighted averaging of an 8x8 data array is illustrated by Fig. 16. Crossed picture elements belong to the first pyramid level above bottom (i=1; 3x3 adjacencies), circled elements to the second level (i=2; 7x7) and the doubled circled one to the third level (i=3; 15x15).

The overlapping and weighted combination of 3x3 neighborhoods serves for the first pyramid level above the pixel level. Higher levels require larger windows

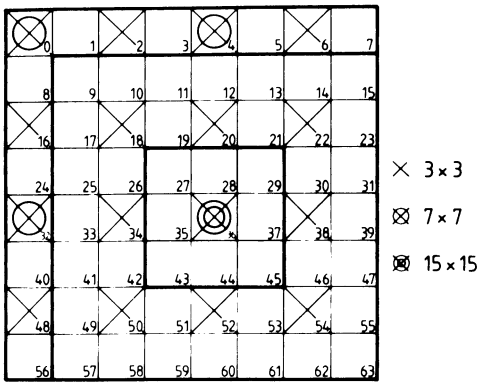


Fig. 16. Bottom layer of 8x8 pyramid: × elements of second layer, ⊗ elements of 3rd layer, ⊗ element of 4th layer with its 3x3 and 7x7 window.

for appropriate overlapping. In order to obtain a digitized approximation to a Gaussian weighting we choose a=2 and b=1. Fig. 17(a) through (c) illustrate the 1-dimensional (horizontal or vertical) overlapping. Starting from the bottom, i.e. pyramid level i=1, each higher level reduces the data set to any other element, horizontally and vertically, each being a weighted average of its $(2^{i+1}-1) \times (2^{i+1}-1)$ neighborhood. Repetition of the two iterations according to Fig. 15(a) enlarges the windows to size $(2^{i+1}+1) \times (2^{i+1}+1)$. Let $m=2^i-1$: To obtain windows of size $(2m+1) \times (2m+1)$ we need 2m iterations, or in other words 2m additional iterations are required to ascend from pyramidal level i to level i+1. We will show that if the constants a and b are chosen as indicated above (a=2, b=1), the weights assigned to the data elements in the window of size $(2m+1) \times (2m+1)$ approach a Gaussian distribution as m approaches infinity. We consider the 1-dimensional case first. According to its definition m is the number of data elements located to the right as well as to the left of the center element in a window. Hence, the width of the window is $2m+1 = 2^{i+1}-1$ (cf. Fig. 16). For the 1-dimensional case the window of width 1 has weight 1. If the width of the window is 3, the weights are: 1/4, 1/2, 1/4 (cf. the signal-flow graph of Fig. 15(a)). Any repeated application of the operation increases the window size by one adjacent element to the right and one to the left. The signal-flow graph applies the weighting operation simultaneously to any two neighbors. It is therefore apparent that the weights are binomial numbers of odd exponent, normalized by 2^{2m}.

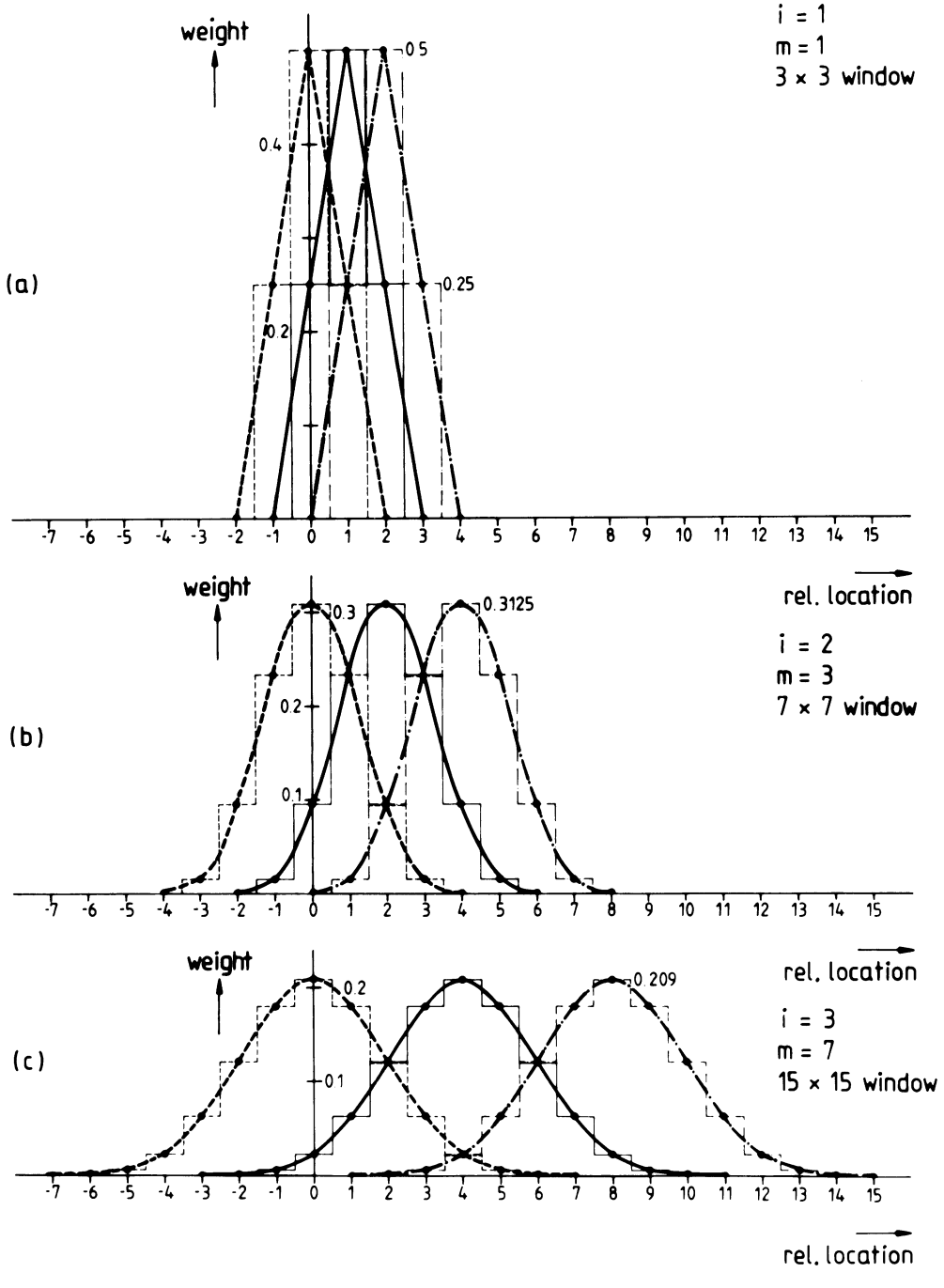


Fig. 17. 1-dimensional weighting and overlapping: (a) window width = 3, (b) w.w. = 7, (c) w.w. = 15.

Hence, the weights are

$$W(D) = 2^{-2m}\binom{2m}{m-D} , \tag{18}$$

where D is the distance of an element from the center of the window. Due to the separability of the transformation into row and column iterations (cf. Fig.15(a)) it is straightforward to extend (18) to the 2-dimensional case. Let D_x and D_y denote the distance of an element from the center element in x and y direction, respectively. Then we have:

$$\begin{aligned} W_{xy} &= (1/2)^{4m} \binom{2m}{m-D_x} \binom{2m}{m-D_y} , \\ &= \frac{(2m)!^2}{2^{4m}\,(m+D_x)!\,(m-D_x)!\,(m+D_y)!\,(m-D_y)!} \end{aligned} \tag{19}$$

Since (18) is equal to the binomial distribution for p=1-p=1/2, it is proven that W(D) approaches a Gaussian shape if m tends to infinity [31]. An example of a Gaussian weighted 5x5 window (m=3; i =2) is shown in Fig. 18 along with a pseudo 3-dimensional view of the weighting window.

$\times 10^{-3}$

0.244	1.465	3.662	4.883	3.662	1.465	0.244
1.465	8.789	21.973	29.297	21.973	8.789	1.465
3.662	21.973	54.932	73.242	54.932	21.973	3.662
4.883	29.297	73.242	97.656	73.242	29.297	4.883
3.662	21.973	54.932	73.242	54.932	21.973	3.662
1.465	8.789	21.973	29.297	21.973	8.789	1.465
0.244	1.465	3.662	4.883	3.662	1.465	0.244

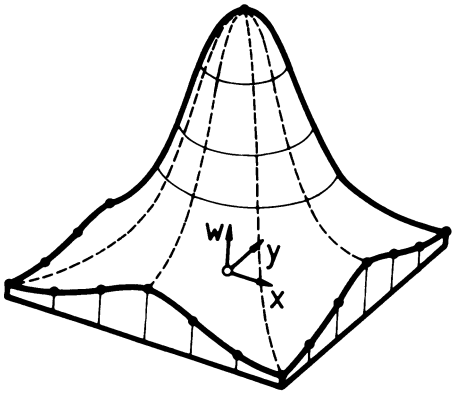


Fig. 18. Gaussian weighting of a 7x7 window.

The complexity of the local convolution is of $O(N^2)$, where N is the length of a row or a column. It has been shown that an inverse signal-flow transform that reconstructs the original data is possible [28]. However, the associated inverse transform matrix need not be sparse. Therefore its complexity may be of $O(N^3)$. Since for 3x3 windows the radix of the signal-flow operation is 3, the number of the operations is $2N^2$. Any enlarging of the window by one element to all the 4 directions accounts for a factor of 2 in the number of operations. Because of the overlapping property, savings due to the reduced number of elements in higher pyramid levels occur only for the last iteration. This is a feature of the Gaussian pyramid which is in contrast to the simple averaging pyramidal transforms that requires only one fourth of the operations for any subsequent level.

4. GLOBAL HIERARCHICAL TRANSFORMATIONS

4.1 Hierarchical Generation of Coefficients

Non-overlapping pyramid generation is an example of global hierarchical transformations. Other choices of the constants a through d in (1) lead to other hierarchical transforms on the basis of the Kronecker product operation (2). The concept of constructing the transform matrix as a Kronecker power of $\mathbf{M}_0$ is a sufficient but not a necessary condition for the existence of hierarchical 2-D radix-2 transformations. In this section we consider the hierarchical generation of transform coefficients and some examples.

It has been observed by several researchers that lexicographical ordering of $2^n \times 2^n$ data arrays allows the calculation of the 2-D Walsh-Hadamard and Fourier transform coefficients by 1-D transformations of order 2^{2n} [32,33]. If used in connection with radix-2 signal-flow graphs this measure does not only simplify the processing procedure, but may also provide a saving in the number of multiplications. Separable transformations are usually computed taking the transformations of rows first, and subsequently transforming the 1-D coefficients columnwise. Signal-flow graphs can be devised so that the first n iterative steps combine the data elements of individual rows, thus yielding their 1-D coefficients. The remaining n iterations serve for combining corresponding 1-D coefficients to 2-D coefficients. For an 8x8 data array the procedure is illustrated by Fig. 19(a).

Dyadic addressing as described in section 2.1 may be viewed as an instruction how to order 2-D data in a vector in such a way that the sequence of elements is that of hierarchically ordered subsquares of size 2x2, 4x4, 8x8, etc. This particular ordering alone does not guarantee the hierarchical generation of 2-D coefficients as a result of an operation according to (3). It is still to be shown under which

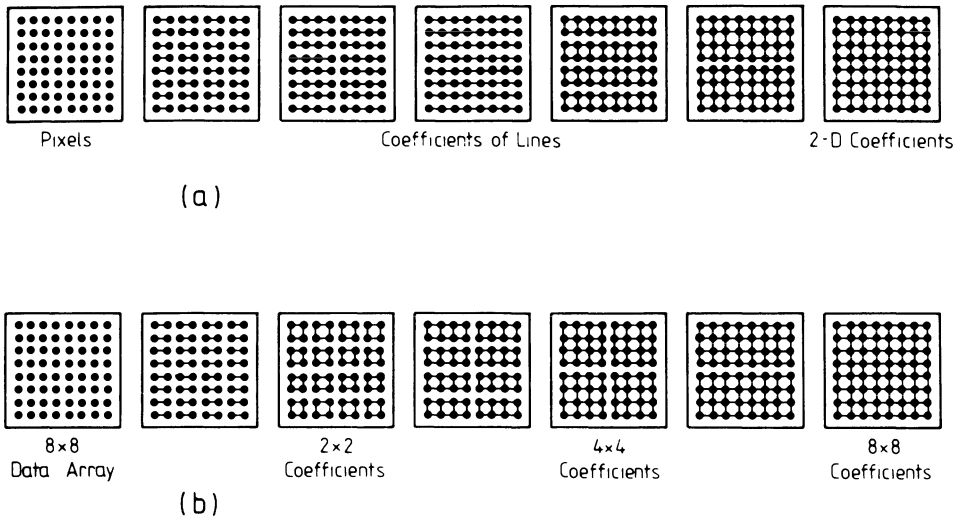


Fig. 19. Combination of 2-D data: (a) row and columnwise, (b) hierarchically.

conditions 1-D radix-2 processing of 2-D data produces the desired 2-D coefficients. It is the aim of this section to show how to generate the 2-D coefficients of a 2-D array (image) from those of subimages. Consider an array of the dimension $2^n \times 2^n$ which is subdivided into subsquares of size $2^{n-1} \times 2^{n-1}$, $2^{n-2} \times 2^{n-2}$, etc. We investigate the conditions required to successively generate the coefficients of all 2x2, 4x4, 8x8, etc. subsquares, each one from the coefficients of its 4 subsquares.

In order to develop signal-flow graphs for hierarchical structures, the first two iterative steps are to combine adjacent data elements to coefficients of 2x2 subsquares: vertically and horizontally by the 1st and the 2nd iteration, respectively. The next two iterative steps combine the coefficients of four adjacent 2x2 subsquares to the coefficients of 4x4 subsquares, etc. Fig. 19(b) depicts an example of hierarchical generation of 2-D coefficients. From (1) through (3) and from Fig. 19(b) the conditions for hierarchical coefficient generation become discernible. The first and the second iterative step, which combine horizontally and vertically adjacent elements must be of equal butterfly structure, i.e. must have equal multipliers and equal combining operators. This condition must hold for any two iterations numbered 2i-1 and 2i, for i=1,2,....,n. For instance, any transformation obtained by Kronecker product expansion meets the above conditions and hence may be computed hierarchically.

4.2 Examples of Global Hierarchical Transformations

Stipulations imposed by hierarchical 2-D transform coefficient generation allow direct application of the scheme to only one orthogonal transformation, viz. the Walsh-Hadamard transform (WHT). For a=b=c=1 and d = -1 the WHT is seen to be a special case of this class of hierarchical 2-D transformations. The WHT coefficients can therefore be obtained in hierarchical order: first all the 2x2 coefficients, from these all the 4x4 coefficients, etc. [25] . Since the coefficients of other hierarchical transformations may be obtained via the WHT, there is another way of generating orthogonal transform coefficients hierarchically [34] .

An important field for application of hierarchical transformations is transform coding of images using variable block size [34,35] . Though the WHT is not an optimal orthogonal transformation in the sense of its decorrelating properties, it is quite popular for its speed and its ease of implementation. We may now add another advantage of the WHT: coefficients may be computed hierarchically. This feature allows adaptive control of the size of transform blocks depending on their spatial frequency contents. Those regions of an image in which elements are highly correlated can be coded most efficiently if the transform blocks are large. Regions of high "business", on the other hand, require spatially denser adaptation, i.e. smaller transform blocks. Applications of hierarchical transformations to gray scale coding and dither processing can be found in [35] and [36] , respectively.

An example of a nonlinear transformation that may be computed hierarchically is the Rapid Transform [37] . Its butterfly structure is similar to that of the WHT except that after each combination in a butterfly the absolute values of the results are taken. This nonlinear transformation has been proved to be invariant under (cyclic) shifts and inversion [37] . Hierarchical or pyramidal transformation serves for detection and rough localizing of objects the dimensions of which are less than half the size of subpictures.

5. PARALLEL ARCHITECTURE FOR PYRAMIDAL TRANSFORMATIONS

Algorithms for both pyramid generation as well as for local or global transformations are based on radix-p signal-flow graphs. Hence, data and operations are of highly regular structure so that spatial parallel computation becomes feasible. In this section we consider parallel processing on the various pyramid levels [22, 23, 38] and macro pipelining of pyramidal operations.

First an architecture is described that supports radix-2 based in-place transformations. The aim is to enable fast execution of the 2-D algorithms. In view of the regular structure of signal-flow graphs it seems obvious to employ single instruction multiple data parallel processing. For this purpose the image data store must be partitioned so that memory sections can be connected to the processors appropriately. Usually this task is performed by static or dynamic single instruction multiple data interconnection networks. For the purpose of supporting radix-2 based in-place algorithms the signal-flow graph itself provides a means for dynamic interconnection. Note that in any iteration of the signal-flow graph only those pairs of memory locations become combined which have Hamming distance 1, (cf. Fig. 2). If a processor was provided for each memory location (pixel) we would obtain a 4^n hypercube machine.

The regular structure of signal-flow graphs may also be exploited so that a smaller number of processors is employed, but none of them remains idle at any time. Consider a signal-flow graph for linewise scanned 2-D data, e.g. Fig. 7. As can be observed from the figure the memory locations may be subdivided into 2^n subgroups (corresponding to the number of lines) which need pairwise processing. This highly desirable property of the signal-flow graph offers an opportunity to have up to 2^{n-1} processors executing identical instructions simultaneously. A dedicated processor can then be assigned to any two memory sections. In section 2.3 it was shown that signal-flow graphs for linewise ordered pictorial data require two iterative steps of radix-2^1, two of radix-2^2, and so forth and finally two of radix-2^n. Fig. 20 displays the steps. A dynamic network is used to connect the sections to the processors as required. An example of a network for connecting an 8x8 array to 4 processors is illustrated in Fig. 21(a). The speed-up factor of parallel processing should ideally be equal to the number of processors, if none of them is idle at any time. The architecture described here approaches

Iteration No.	1	2	3	4	• • •	2n-1	2n
Radix	2^1	2^1	2^2	2^2	• • •	2^n	2^n
Max. No. of Memory Sections	2^{2n-1}	2^n	2^{2n-2}	2^n	• • •	2^n	2^n
Locations per Section	2^1	2^n	2^2	2^n	• • •	2^n	2^n
Max. No. of Processors	2^{2n-1}	2^{n-1}	2^{2n-2}	2^{n-1}	• • •	2^n	2^{n-1}

Fig. 20. Number of memory sections and possible number of processors.

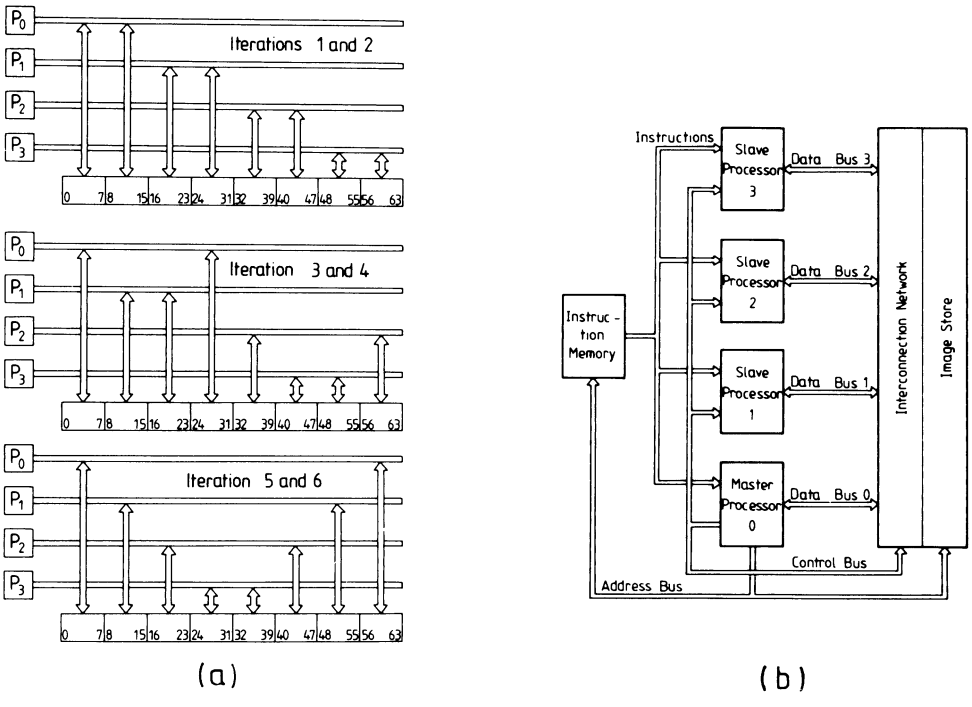


Fig. 21. (a) Memory section allotment for an 8x8 array,
(b) 4 processor systems serving for one pyramid level.

this limit closely. The maximum number of processors would theoretically be 2^{n-1}. But the practical limit may be lower due to the increasing complexity of the interconnection network. An example using 4 microprocessors (MC 68 000) has been realized in hardware. Fig. 21(b) shows the basic structure of this experimental system. It is still a matter of continuing research to optimize the number of processors and their wordlength with respect to throughput and cost.

Parallelism involved in this architecture exploits the possibility of parallel processing in 2-D space. Another form of parallelism is that of multiprocessing in time. This form of operation is usually referred to as pipelining. The latter may be incorporated in the 2-D parallel architecture in order to exploit both space and time for a higher degree of parallelism. Macro-pipelining may be used either for multilevel operation or for multiresolution processing. A special case of the latter is the pyramid architecture depicted by Fig. 22. Connection between the pyramid levels is via the signal-flow graphs, i.e. higher level elements are calculated from the lower levels and stored in their respective storage area. Once all

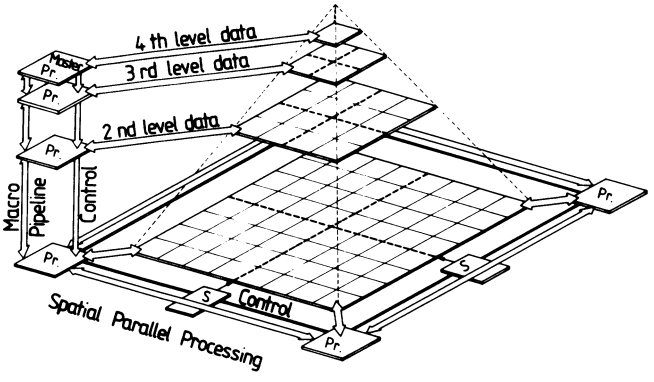


Fig. 22. Example of a pyramidal storage system with processors.

pyramid levels required have been calculated and stored, further operations on the levels may be executed in parallel or pipelined if applicable.

REFERENCES

[1] A. P. Reeves, "Parallel computer architectures for image processing", (Survey), Computer Vision, Graphics, and Image Processing, vol. 25, 68-88, 1984.

[2] K. Preston and L. Uhr (Eds.), Multicomputer and Image Processing: Algorithms and Programs . New York: Academic Press, 1982.

[3] S. L. Tanimoto and A. Klinger (Eds.), Structured Computer Vision: Machine Perception Through Hierarchical Computation Structures. New York: Academic Press, 1980, Chapter 2: S.L. Tanimoto, "Image Data Structures".

[4] A. Rosenfeld (Ed.), Multiresolution Image Processing and Analysis. Berlin, Heidelberg, New York, Tokyo: Springer-Verlag, 1984.

[5] A. Rosenfeld, "Quadtrees and pyramids for pattern recognition and image processing", Proc. 5th Internat. Conf. on Pattern Recognition, Miami Beach, FL, pp. 802-811, 1980.

[6] L. Uhr, "Converging pyramids of arrays", Proc. IEEE Workshop on Computer Architecture for Pattern Analysis and Image Data Base Management, Hot Springs, VA, pp. 31-34, 1981.

[7] M. Shneier, "Two hierarchical linear feature representations: Edge pyramids and edge quadtrees", Computer Graphics and Image Processing, vol. 17, pp. 211-224, 1981.

[8] C. R. Dyer, "Pyramid algorithms and machines", in [2] , pp. 409-420.

[9] S. L. Tanimoto,"Template matching in pyramids", Computer Graphics and Image Processing, vol. 16, pp. 356-369, 1981.

[10] S. L. Tanimoto, "Algorithms for median filtering of images on a pyramid machine", in Computing Structures for Image Processing (M. J. B. Duff, Ed.). London: Academic Press, 1983.

[11] A.R. Gangolli and S. L. Tanimoto, "Two pyramid machine algorithms for edge detection in noisy binary images", Information Processing Letters, vol. 17, pp. 197-202, 1983.

[12] J.-L. Basille and S. Castan, "Multilevel architectures for image processing: Theoretical study and simulation results", Proc. SPIE 2nd Internat. Techn. Symp. on Optical and Electro-Optical Applied Science and Engineering, Cannes, France, vol. 596, Dec. 1985.

[13] P. A. Sandon, "A pyramid implementation using a reconfigurable array of processors", Proc. IEEE Workshop on Computer Architecture for Pattern Analysis and Image Data Base Management, Miami Beach, FL, pp. 112-118, 1985.

[14] T. A. Rice and L. H. Jamieson, "Parallel processing for computer vision", Integrated Technology for Parallel Image Processing, (S.Levialdi, Ed.) pp. 57-78. London: Academic Press, 1985.

[15] G. Fritsch, "General purpose pyramidal architectures", this volume.

[16] C. R. Dyer, "A VLSI pyramid machine for hierarchical parallel image processing", Proc. Internat. Conf. on Pattern Recognition, Dallas, TX, pp. 381-386, 1981.

[17] V. Cantoni, M. Ferretti, S. Levialdi and F. Maloberti, "A pyramid project using integrated technology", in Integrated Technology for Parallel Image Processing (S. Levialdi, Ed.), pp. 121-132. London: Academic Press, 1985.

[18] V. Cantoni, "Pyramidal systems: architectural features", this volume.

[19] S. L. Tanimoto, "Programming techniques for hierarchical parallel image processors", in [2], pp. 421-429.

[20] M. J. B. Duff and S. Levialdi (Eds.), Languages and Architectures for Image Processing. London: Academic Press, 1981.

[21] V. Di Gesu´, "A high level language for pyramidal architectures", this volume.

[22] Ph. W. Besslich, "Algorithmic pyramid machine for image processing", Proc. 18th Annual Internat. Conf. on System Sciences, Honolulu, HI, pp. 202-212, 1985.

[23] Ph. W. Besslich, "Parallel architecture for line-scanned images", Proc. SPIE 2nd Internat. Techn. Symp. on Optical and Electro-Optical Applied Science and Engineering, Cannes, France, vol. 596, Dec. 1985.

[24] A. Klinger and M. L. Rhodes, "Organization and access of image data by areas", IEEE Trans. Pattern Analysis and Machine Intelligence, PAMI-1, pp. 50-60, 1979.

[25] Ph. W. Besslich, "Hierarchical generation of 2-D data structures", Proc. Symp. Mustererkennung 1984, Graz, Austria, pp. 177-183, Sept.1984.

[26] Ph. W. Besslich, "A method for the generation and processing of dyadic indexed data", IEEE Trans. Computers, C-32, pp. 487-493,May 1982.

[27] Ph. W. Besslich, "On radix-2 similarity transformations based on the two-dimensional WHT", Proc. IEEE 1983 Internat. Symp. on Electromagnetic Compatibility, Arlington, VA, pp. 532-556, Aug. 1983.

[28] Ph. W. Besslich, and J. Kurowski, "Globale und lokale Gewichtstransformationen zur Cluster-Analyse", Berichte Elektrotechnik, Universität Bremen, FB-1, ISSN 0724-1933, June 1983, 1/83 (in German).

[29] P. J. Burt, "Fast algorithms for estimating local image properties", Computer Graphics and Image Processing, vol. 21, pp. 368-382, 1983.

[30] P. J. Burt, "The pyramid as a structure of efficient computation", in [4], pp. 6-35.

[31] W. Feller, An Introduction to Probability Theory and its Applications, vol. 1, New York: John Wiley and Sons, 1968.

[32] B. Arazi, "Two-dimensional digital processing of one-dimensional signal", IEEE Trans. Acoustics, Speech, and Signal Processing, ASSP, pp. 81-86, 1974.

[33] R. M. Mersereau and T. C. Speake, "A unified treatment of Cooley-Tukey algorithms for the evaluation of the multidimensional DFT", IEEE Trans. Acoustics, Speech, and Signal Processing, ASSP-29, pp. 1011-1017, 1981.

[34] B.G. Kashef and A. Habibi, "Direct computation of higher-order DCT coefficients from lower-order DCT coefficients", Proc. Applications of Digital Image Processing VII, SPIE Symp. San Diego, CA, pp. 425-431, 1984.

[35] M. F. Carlsohn, "Local resolution enhancement in transform coding and dither binarization of images", IEEE MELCON '85, vol. II: Digital Signal Processing, A. Luque, A.R. Figueiras Vidal, V. Capellini (Eds.), Elsevier Science Publ. B. V. (North Holland), pp. 249-252, 1985.

[36] M. F. Carlsohn, and Ph. W. Besslich, "Adaptive selection of threshold matrix size for pseudogray rendition of images", Optical Engineering, vol. 24, pp. 655-662, 1985.

[37] H. Reitboeck and T. P. Brody, "A transformation with invariance under cyclic permutation for applications in pattern recognition", Inf. Control, vol. 15, pp. 130-154, 1969.

[38] Ph. W. Besslich and J. O. Kurowski, "Quadrant architecture for fast in-place algorithms", Proc. IEEE Workshop on Computer Architecture for Pattern Analysis and Image Data Base Management, Pasadena, CA, pp. 26-33, 1983.

OVERLAPPING IN COMPACT PYRAMIDS

M. Ferretti
Dipartimento di Informatica e Sistemistica
Universita' Statale di Pavia
27100 Pavia, Italy

I. Introduction.

Hierarchical methods are well established in computer vision. The idea of processing images at various level of resolution is motivated by many factors, such as decreased computational cost, feasibility of divide-and-conquer methods, interaction of region based and point based operations. As a consequence, hierarchical data structures, broadly defined "pyramids", have been introduced to experiment such ideas; among the various definitions of these data structures, overlapped pyramids have gained increasing interest especially for segmentation purposes. Meanwhile, the advantages of multi-resolution techniques have lead to the proposal of augmenting mesh-connected massively parallel architectures with pyramid like topology; a few such machines, designed to operate in SIMD or multi-SIMD mode, are in advanced stage of design or realization. It seems therefore interesting to study the effectiveness of pyramid architectures in some multiresolution algorithms, especially when the match between architecture and data structure is only partial; such is the case when machines with fixed connections among neighboring processing elements are used to run algorithms which exploit overlapped pyramids with dynamic links among adjacent levels.

In the following, section II briefly reviews the various definitions of pyramids with special focus on the degree of overlapping and on reduction of resolution; also it contains

NATO ASI Series, Vol. F 25
Pyramidal Systems for Computer Vision
Edited by V. Cantoni and S. Levialdi
© Springer-Verlag Berlin Heidelberg 1986

a short description of compact pyramids architectures which are best suited to VLSI implementation due to their regular topology; section III analyses the efficiency of such machines in embedding overlapped pyramid data structures especially from the point of view of data movements; section IV introduces the linking technique exploited with overlapping in segmentation algorithms and studies the problem of maintaining dynamic links in compact pyramids; finally, section V evaluates possible architectural improvements of such machines to enhance their performance in those tasks which do require overlapping and/or linking.

II. Pyramids Definitions

Many varieties of hierarchical representations of images, or image related properties, have been proposed: processing cones introduced by Uhr [1], pyramids as defined by Tanimoto and Pavlidis [2] and Klinger and Dyer [3], recognition cones by Hanson and Riseman [4] and overlapped pyramids, a class to which Hanson and Riseman structures belong, but which has received much attention by many authors [5]-[10]. All these representations consist of a stack of images of reduced resolution in which each pixel of a level represent a block of pixels in the level below down to the original image. A unifying approach to the classification of all versions of such data structures can be based on three features: tessellation topology, width of the support upon which successive levels are build (referred to as span in the following) and degree of reduction between adjacent levels.

As far as tessellation is concerned, the most common solution adopted is that of square subdivision; alternative tessellations have been proposed that partition the plane into arrays of equilateral triangles or regular hexagons [11]. While such topologies offer some advantages over square tessellation in terms of discrete approximation of straight

lines, they are less suited to the construction of uniformly scaled hierarchies.

The choice of span and of reduction introduces a partition of these data structures into non-overlapped and overlapped pyramids. Non-overlapped (basic) pyramids are obtained if span and reduction are equal and both integers. It has become common the definition of odd/even pyramids according to parity of span. Fig. 1 gives two examples of basic pyramids in one dimension, the former having span and reduction equal to 2, the latter equal to 3.

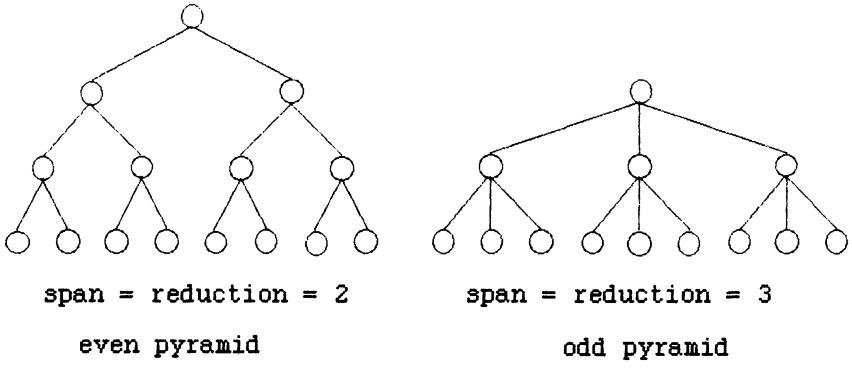
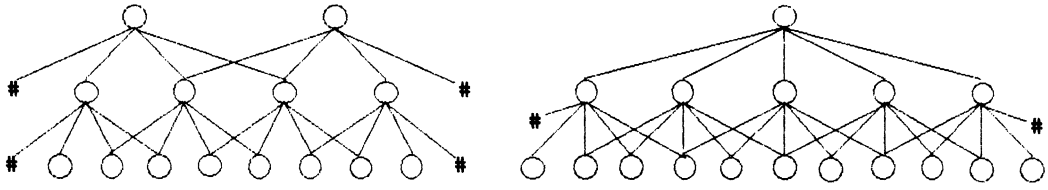


Fig. 1

The two dimensional extension of the first pyramid is the data structure most commonly referred to as basic pyramid (provided that it is augmented with explicit connections among pixel of the same level). It has been used extensively for many numerical and seminumerical algorithms [12] (bit counting, area and perimeter calculation, histogramming, sorting, median filtering) and has become a test bed for the efficiency of the pyramid in many tree related problems [13]. Yet, the sharp subdivision of the image into 2x2, non overlapping blocks causes major difficulties to most local image processing operations, such as edge detection.

Overlapped pyramids are obtained if span is greater than reduction. Fig. 2 gives one dimensional examples of an even

pyramid (span=4, reduction=2) and of an odd one (span=5, reduction=2). In these data structures the blocks of pixels associated to pixels of reduced resolution level overlap to a degree which depends on both span and reduction. The two pyramids obtained as extension of the one dimensional examples of Fig. 2 give rise respectively to 50% overlapping in lateral directions and to 25% in diagonal direction and to 60% and 36%.



span = 4 reduction = 2

even pyramid

span = 5 reduction = 2

odd pyramid

Fig. 2

Many are the advantages which result from overlapping: broadly stated, the properties computed at higher levels in the pyramid are less sensitive to uncertainties arising from the limited support on which they are built. For examples, curves can be represented by line segments detected with a local operator at the image level and good continuation can be guaranteed by appropriate processing at higher levels, since adjacent segment representations in these levels share a set of edge-pixels in levels below. Furthermore, overlapping can be exploited to build intensity pyramids which are the equivalent of convolving the image with a set of Gaussian-like kernels differing in size by an amount equal to reduction, but all alike in shape (these pyramids are now commonly known as Gaussian pyramids) [14],[15].

Another consequence of overlapping is that each pixel in a given layer contributes to more than one pixel in the level above in the process of building the pyramid. It becomes therefore possible an iterative process of constructing the

pyramid: after an initial step in which the property is computed at each level above the image plane on the base of the whole set of pixels which form the kernel, each pixel can be made to contribute to that pixel in the level above (among the so called candidate fathers) which is most similar to it, according to a similarity criteria. This introduces a linking mechanism which has lead to some new, very powerful computational techniques. The experience gained with this approach has lead to variations to the linking mechanism: the initial forced linking strategy has been replaced by unforced linking [6],[8],[10],[16], which introduces in the pyramid a forest of trees, whereby a node in the pyramid can be a root (the apex of a subtree at a level below the highest one), a leaf (a node in mid-level not linked to any of its offsprings in level below) or a normal node, having both offsprings and a father. An alternative to the forced-unforced dichotomy is dynamic weighted linking [17]: in this technique the contributions of a node to its fathers (weights) are dynamically adjusted in iterative fashion.

It is not the purpose of this paper to discuss the relative advantages of these techniques; however, substantial evidence exists to say that they are a good set of tools for multiresolution processing. It would be therefore interesting that pyramidal architectures allow for an easy implementation of such algorithms, even when the data structures they use matches only partially such architecture; and this happens with all kinds of overlapping and linking.

Compact pyramids architectures are discussed in details in other parts of this book. For the purpose of this paper, it suffices to say that such architectures [18-20] are the hardware implementation of the basic, non overlapped pyramid data structure, augmented with connections among neighboring PEs in each level. All of the interconnections are bit serial and the transfer time to transmit a value between two adjacent PEs shall be called in the following time unit. Such machines are generally operated in SIMD mode by a single controller, which distributes to all PEs the instruction to be executed. Memory references and near-neighbor selection

are therefore homogeneous in the whole machine. Also, each PE is equipped with a condition register which enables/disables it from processing; the setting of this register is established globally by the controller or is the result of a local operation.

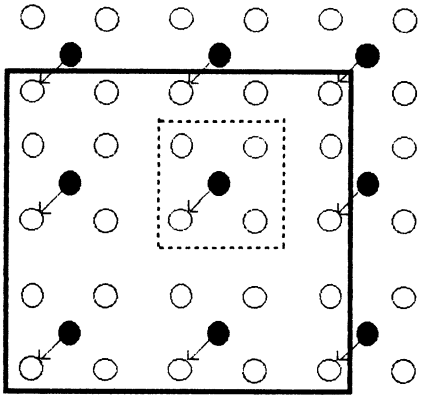
III. Efficiency of Compact Pyramids

The analysis of the efficiency of compact pyramids architectures in simulating overlapped data structures requires a preliminary discussion of the association of PEs to pyramid nodes.

Since the reduction factor in the number of PEs of adjacent levels is 4, no problems arise if the pyramid to be simulated exhibits a reduction equal to this figure. Also, reduction factors which are a multiple of 4 can be easily accommodated by letting a fraction of PEs equal to this multiple work, while the rest stay idle: this can be obtained preliminarily disabling such PEs by means of the condition register. Conversely, if the pyramid has a number of nodes in each successive level higher than the number of PEs available (see for example the interleaved pyramid [21]), it is necessary to split computations into more cycles, time sharing each PE to perform the functions of the lacking nodes. These situations are however rather uncommon.

More important is the the problem of how to deal with odd pyramids in an architecture which is inherently even. For example, gaussian pyramids are best defined over a kernel of span 5x5; the central node of this kernel has a father which is logically located above it, while PEs in compact pyramids are centered in the middle of the square formed by their sons. Nothing however prevents a father PE from being considered logically placed on top of one of its sons; for example Fig. 3 shows the mapping of PEs to nodes in a pyramid built with a kernel of span 5x5.

The crucial factor in the analysis of the efficiency of compact pyramids is the speed of access to the values of the neighborhood. Independently from the property which is computed at each node, it is important that the architecture does not introduce delays during the process of collection of the values of the neighborhood necessary to each PE. The cost of such movements shall be calculated both for bottom-up (collection phase) and for top-down (distribution phase) processing.



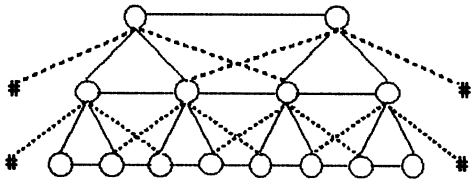
PEs to nodes association

Fig. 3

The analysis of data movement technique shall be done initially for the 1-D case. Fig. 4 shows a subset of a compact 1-D pyramid, augmented with logical connections (dashed lines) among fathers and logical sons PEs (those not directly connected) which show the complete kernel of the data structure to be simulated (span=4, reduction=2).

It is straightforward to see that collecting the property values of the whole set of sons, including candidate ones, requires exactly span time units. Indeed, since the processing mode is SIMD, each father PE collects the values of its direct sons using the existing hardware links (two

time units); then each father fetches from its two neighboring PEs in its own level the remaining two values of its logical sons, one from its East neighbor, the other from the West one (two more time units). The distribution phase is split into three steps (three time units): in the first one, each couple of direct sons access the property value of their common father, in the second one even sons fetch their second father value from the west neighbor, while odd sons stay idle (the condition register is set at odd locations) and in the third step even and odd sons interchange role. Therefore top down distribution takes span-1 time units. This figures, obtained with span=4, are actually general for 1-D pyramids, as can be easily shown. From the point of view of SIMD processing it is extremely important that values collected in either phase get stored in local memory at common addresses, so that SIMD processing can proceed at full speed in subsequent operations. Actually this happens no matter how large is the kernel.



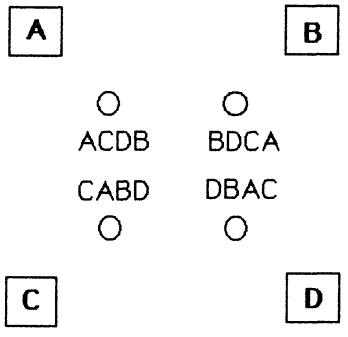
1-D overlapped pyramid

Fig. 4

The extension of the analysis to the 2-D case shall be done for the even pyramid of span 4 and reduction 2 for sake of simplicity; the results are however easily extended to other spans, for example to the gaussian pyramid with 5x5 kernel.

With considerations analogous to the 1-D case, it can be shown that the collection phase takes exactly 16 time units (that is span x span): thus the compact pyramid introduces no

delay even if it does not embed explicit overlapping. The distribution phase requires a detailed description. In this situation, each PE has four candidate fathers in the level above and the minimum amount of time units necessary to collect their values is therefore 4. But this figure cannot be obtained; indeed, it turns out that this phase requires (2N)-1 steps, if N is the number of candidate fathers. To show this, let us consider Fig. 5. It depicts a small section of the pyramid containing four father PEs (denoted with capital letters in square boxes), the four PEs in the level below (denoted with empty circles) that share them as candidate fathers and the final situation after the whole process (fathers values below each son).



Top-down distribution

Fig. 5

The steps necessary to distribute to each son all of the values of its candidate fathers consist of a top down propagation and of a distribution phase in the sons plane. This second phase requires 2(N -1) time units and can be accomplished in two ways. In both cases, candidate father values are collected at the sons in synchronized fashion ("own" father first, then "column", "diagonal" and "row" one or "column", "row" and "diagonal" father) and can be stored in local memory at common addresses. The performance of the compact pyramids suffers considerably in this situation; in

section V the architectural features responsible for this slow down shall be discussed and the cost of possible improvements evaluated.

So far, it has been implicitly assumed that the pyramid property can be evaluated at each PE only after collecting all of the values of candidate sons. It would be nice to be able to split the computation leading to the property value in such a way that partial results can be distributed among neighboring PEs, instead of simply transmitting the sons values. One might think that symmetry and separability of the kernel guarantee such improvement; unfortunately this is not the case, as can be easily verified for example with gaussian kernels. The limit situation of a uniform kernel (simple averaged pyramid) however partially benefits from this idea: the naive processing mode requires 16 units for transfers, 15 units for sums and a last unit for rescaling, while it is possible to execute two sums immediately after collecting direct sons values and to distribute them to PEs of the same row (or column), which diminishes by two the steps required for transfers and sums (a slightly different result can be obtained if the compact pyramids has eight connectivity in the plane).

IV. Maintaining Dynamic Links

As explained in section II, powerful segmentation techniques have been developed which use overlapped pyramids with dynamic reconfigurable connections (links) among father nodes and candidate sons. After an initialization phase, each son examines the property values of its candidate fathers and decides to which one (if any) it shall contribute in successive steps; the fathers themselves recompute their value on the base of the subset of their candidates sons they are linked to. This process is guaranteed to converge to a final situation which segments the image in regions of

homogeneous property. The pixels in the image plane receive such a value from the apex of the subtree they belong to (apex nodes distribute their final value only to linked sons, down to the base).

Maintaining this dynamic structure of connections in a compact pyramid requires adding some bits to the values transmitted upwards, to encode the link between a PE and the chosen father PE ; the amount of extra bits is of course sup(log N). Indeed, because of SIMD processing it is necessary to perform the complete collection phase before re-estabilishing the property values of PEs in higher level; even if a father is linked to a subset of its candidate sons, this subset is different in principle at each node. Therefore no reduced sequence of near-neighbor addresses can be established a priori by the controller of the pyramid.

Moreover, each PE must keep the code which represents the father it is linked to. This is not necessary during the process of stabilization of links, it is unavoidable to choose among the values distributed during the final top-down segmentation phase.

A minor modification of this coding scheme is needed in the unforced version of linking. PEs marked as leafs simply receive and transmit data for the benefit of neighboring PEs during the collection phase, but do not recompute their property value, being disabled at that stage by the condition register. Root PEs send out their value with a code which signals receiving PEs that such datum must not be used; and they also get disabled together with leaf ones.

Other versions of unforced linking dynamically adjust the strength of links among sons and fathers; here, no coding is required, because all of the values of sons are used, but the link strengths must be transmitted along with the values. Coding is nevertheless reintroduced in the segmentation phase; root nodes are identified on the basis of the sum of their link strengths to fathers (a negligible value in this sum means that the node is not linked to nodes above its level) and non-root nodes are connected to the most strongly linked father.

V. Architectural improvements

The analysis carried out so far has shown that compact pyramids are quite efficient in simulating overlapped data structures, except for top-down data movements.

The inefficiency in this situation does not depend on the pyramid topology, because the extra time units required to distribute fathers values to candidate sons are spent while processing takes place in the sons plane. The slow-down is caused by the unidirectional connections among PEs and cannot therefore be eliminated at low cost.

As far as linking is concerned, one might expect better performances in compact pyramids if links could be explicitly maintained by means of an "indexing near-neighbor" vector, that is a set of flags indicating to which father a node is linked to. But this facility requires that each PE is physically connected to its candidate fathers, otherwise it is useless; such connections increase greatly the complexity of the architecture (this would be a truly overlapped machine). The solution adopted here (coding) is much more flexible and does not degrade the performances of the machine too much.

The conclusion which can be drawn is that the basic compact pyramid allows an easy implementation of overlappingand linking in an architecture whose complexity is kept at a minimum.

REFERENCES

[1] L. Uhr, "Layered recognition cones, networks that preprocess, classify and describe", IEEE Trans. on Computers, vol. 21, pp. 756-768, July 72

[2] S.L. Tanimoto, T. Pavlidis, "A hierarchical data structure for picture processing", Computer Graphics and Image Processing, vol. 4, pp. 104-109, 1975

[3] A. Klinger, C.R. Dyer, "Experiments on picture representation using regular decomposition", Computer Graphics and Image Processing, vol. 5, pp. 68-109, 1976

[4] A. Hanson, E. Riseman, "Segmentation of natural scenes", in Computer Vision Systems, A. Hanson and E. Riseman Eds., New York, Academic Press, 1978

[5] P. Burt, T.H. Hong and A. Rosenfeld, "Segmentation and estimation of image region properties through cooperative hierarchical computation", IEEE Trans. Syst. Man Cyb. , vol. 11, pp.802-804, 1981

[6] H. J. Antonisse, "Image segmentation in pyramids", Computer Graphics and Image Processing, vol. 19, pp. 367-383, 1982

[7] T. Silberberg, S. Peleg and A. Rosenfeld, "Multi resolution pixel linking for image smoothing and segmentation", in Proc. Image Understanding Workshop, Washington, D.C., pp.32-38, 1981

[8] M. Pietikainen and A. Rosenfeld, "Image segmentation by texture using pyramid node linking", IEEE Trans. Syst. Man Cyb., vol. 11, pp. 822-825, 1981

[9] M.O. Shneier, "Extracting linear features from images using pyramids", IEEE Trans. Syst. Man Cyb., vol. 12, pp. 569-572, 1982

[10] T.H. Hong and A. Rosenfeld, "Unforced image partitioning by weighted pyramid linking", in Proc. Image Understanding Workshop, Palo Alto, CA, pp. 72-78, 1982

[11] S.L. Tanimoto, J.P. Crettez and J.C. Simon, "Alternative hierarchies for cellular logic", in Proc. 7th Intern. Conf. on Pattern Recognition, Montreal, Canada, pp. 236-239, 1984

[12] S.L. Tanimoto, "Sorting, histogramming and other statistical operations on a pyramid machine", in Multiresolution Image Processing and Analysis, A. Rosenfeld Ed., Springer Verlag, 1984

[13] Q.F. Stout, R. Miller, "Data movement techniques for the pyramid computer", to be published

[14] P.J. Burt, "Fast filter transforms for image processing" in Computer Graphics and Image Processing, vol. 16, pp.

20-51, 1981

[15] P.J. Burt, "The pyramid as a structure for efficient computation", in Multiresolution Image Processing and Analysis, A. Rosenfeld Ed., Springer Verlag, 1984

[16] J. Cibulskis and C.R. Dyer, "Node linking strategies in pyramids for image segmentation", in Multiresolution Image Processing and Analysis, A. Rosenfeld Ed., Springer Verlag, 1984

[17] T.H. Hong and A. Rosenfeld, "Compact region extraction using weighted pixel linking in a pyramid", IEEE Tran. on Pattern Anal. and Mach. Intell., vol. PAMI-6, pp. 222-229, 1984

[18] S.L. Tanimoto, "Towards hierarchical cellular logic: design considerations for pyramids machines", Comp. Sc. Dept., Univ. Washington, Techn. Report 81-02-01, Feb. 81

[19] V. Cantoni, M. Ferretti, S. Levialdi and R. Stefanelli, "PAPIA, Pyramidal Architecture for Parallel Image Analysis", IEEE Comp. Arch. Newsletter, pp. 14-25, 1985

[20] D.H. Schaefer, G.C. Wilcox and V.J. Harris, "A pyramid of MPP processing elements- experiences and plans", Proc. 18th Annual Hawaii International Conference on System Sciences, Vol. 1, pp.178-184.

[21] N. Ahuja and S. Swamy, "Multiporcessor pyramid architectures for bottom-up image analysis", in Multiresolution Image Processing and Analysis, A. Rosenfeld Ed., Springer Verlag, 1984

SOME PYRAMID TECHNIQUES FOR IMAGE SEGMENTATION

Azriel Rosenfeld
Center for Automation Research
University of Maryland
College Park, MD 20742

ABSTRACT

This paper describes a collection of multiresolution, or "pyramid", techniques for rapidly extracting global structures (features, regions, patterns) from an image. If implemented in parallel on suitable cellular pyramid hardware, these techniques require processing times on the order of the logarithm of the image diameter.

1. Introduction

One of the most important potential applications of pyramid machines is the fast detection and extraction of global structures (features, regions, patterns, etc.) in an image, by rapidly combining information collected from many parts of the image. A number of pyramid-based methods of extracting global image structures have been developed at our laboratory, and others have been proposed, as described below.

Fast detection of global structure in an image seems to be an essential component of real-time perception. Humans are able to recognize objects in unexpected, complex images in one second or less, a period of time during which only on the order of 100 neural computational steps could have taken place. Obviously, the human visual system is highly parallel, but conventional parallel processing concepts are not powerful enough to account for this performance. For example, a human can "immediately" detect a long straight line in the visual field; since the line may be several hundred retinal receptor cells ("pixels") long, conventional methods would require several hundred computational steps, even on a two-dimensional parallel machine (cellular array or mesh), to extract the line from the input image. Pyramid-style parallelism provides a much faster method of doing this; as we shall see below, a pyramid scheme can extract long straight lines from an image in tens, rather than hundreds, of steps. Schemes of this type seem to be essential in achieving fast recognition of global patterns. Such patterns cannot be reliably recognized

NATO ASI Series, Vol. F 25
Pyramidal Systems for Computer Vision
Edited by V. Cantoni and S. Levialdi
© Springer-Verlag Berlin Heidelberg 1986

using conjunctions of local features; some means of explicitly extracting global structures is needed.

The grouping algorithms discussed here will be primarily two-dimensional, but similar pyramid-based techniques can be applied to the analysis of images of three-dimensional scenes, or to time-varying images. These extensions will be briefly discussed at the end of the paper.

2. Bottom-up techniques: Intensity-band

In this section we describe methods of feature detection and texture analysis based on pyramid processing of arrays of intensity data (gray levels or colors) or local property maps derived by applying feature detection operators to intensity arrays. Another class of techniques, based on pyramid processing of contour data (edges or curves), will be discussed in the next section.

2.1. Summarizing intensity data: Modality detection

It is well known that a simple pyramid algorithm can count the number of occurrences of any given value in an image, using a "divide-and-conquer" technique, in time proportional to the height of the pyramid, i.e., to the log of the image diameter. In order to compute the histogram of the image, this algorithm must be repeated as many times as there are distinct values. Humans do not seem to be able to compute histograms effectively; a mixture of three values is hard to distinguish from a uniform distribution [1]. On the other hand, humans are good at detecting bimodality; this may underlie the phenomenon of "similarity grouping" in perception [2]. In the following paragraphs we describe a pyramid-based algorithm that directly detects bimodality (in the values of some given local property) without computing a histogram.

Our scheme is based on a simple pyramid structure in which each cell at level h receives inputs from a block of cells (its "children") at level $h-1$. The image is initially input to the base of the pyramid (level 0), and the given local property is computed (wherever applicable), yielding a (sparse) array of local property values.

Each cell at level 1 now examines the set of values in its block of the (level-0) array, and finds the partition of these values into two subsets that minimizes the variance of each subset about its mean. (This partition is sometimes called the "bimean" [3].) Let the means of the subsets be μ and ν, let the sizes of the subsets

be r and s, and let the standard deviations be σ and τ. The cell then computes the "Fisher distance" $d \equiv |\mu - \nu|/\sqrt{\sigma^2 + \tau^2}$ between the two subsets. Note that a large Fisher distance means that the population of values in the cell's block is strongly bimodal (the means are many standard deviations apart). If the Fisher distance is not sufficiently large, the cell flags itself as "not bimodal".

Each cell on level 2 now examines the data computed by its "children" on level 1. Suppose that all (or nearly all) of the children have bimodal populations, say with means and standard deviations $\mu_1, \nu_1, \ldots, \mu_m, \nu_m$ and $\sigma_1, \tau_1, \ldots, \sigma_m, \tau_m$. The cell finds the partition of the μ's and ν's into two subsets that minimizes the variance of each subset about its mean. (The means of the subsets can be computed as weighted averages from the μ's and ν's and their corresponding r's and s's; the variances can be estimated from the μ's, ν's, r's, s's, σ's and τ's.)

Finally, the cell estimates the Fisher distance for this new partition, and decides whether its combined population of values is still bimodal. If several of the cell's children do not have bimodal populations, the cell does not carry out these computations, but simply flags itself as "not bimodal".

This process is repeated for the cells on levels 3, 4, $\cdots$. We suggest that bimodality is "perceptually conspicuous" if a cell on a sufficiently high level (representing a large block of the image) has a bimodal population.

In principle, this scheme could be generalized to detect trimodality, . . . , k-modality, . . . , but the amount of computation that the cells would need to perform would grow rapidly with k. In any case, since our method is based on repeatedly estimating the variances of the subsets, it is not capable of fine discriminations between subsets whose means are close together; thus the method will detect bimodality only when the means are very far apart or the variances are very small, and there would be little value in attempting to detect k-modality for $k > 2$. [In general, it would be of little value for the brain to perform computations whose results would be so inaccurate as to provide no useful information. Note that if the brain contains pyramid-like structures, they would not be perfectly regular (i.e., the image blocks seen by the cells would not all be of equal sizes nor have equal degrees of overlap); thus it would be pointless to attempt to use them, e.g., for exact counting, but only for rough population size estimation.]

Detection of "anomalies" can be regarded as a special case of bimodality detection. If there is a unique value that differs considerably from all the other values, it defines a subpopulation that has high Fisher distance from the rest of the population, so that we have strong bimodality.

2.2. Summarizing intensity data: Polynomial fitting and feature detection

It has been shown [4] that a pyramid can recursively compute least-squares polynomial fits to the blocks of image data underlying its cells. We shall consider here only the case where the polynomials have degree 0 (note that the best-fitting constant to a set of values is the mean, and the average squared error of this fit is the variance), but higher-degree fits might also be of interest, e.g., in segmentation using Haralick's facet-model approach.

We can use this method of polynomial fitting to detect compact homogeneous regions in an image, where "homogeneous" means "a good fit to a polynomial of the given degree" (so that in the degree-0 case it means "low-variance"). To do this, each cell on level 1 computes the mean and variance of the values in its block, and this process is repeated at levels 2, 3, $\cdots$. Let P be a cell on level h, and let $Q_1, \ldots, Q_k$ be the cells on level $h+1$ whose blocks of the image contain P's block. Let σ be the variance of P, and $\tau_1, \ldots, \tau_k$ the variances of the Q's; let the area of P's block be s, and the areas of the Qs' blocks be t. We call P a "root node" if $\tau_1/t, \ldots, \tau_k/t$ are all significantly higher than σ/s. Intuitively, this will happen if P's block lies (mostly) within a "homogeneous" region of the image (e.g., a region whose values are independent samples of the same normal distribution), but the Qs' blocks are too big to lie inside that region. Note that for low levels h, if the image is noisy, this criterion will not be reliable, because the τ's may be too variable to be good estimates of the variability of the region's population of values; but for larger values of h, the estimates will be more reliable, and τ_i/t will be smaller than σ/s (since t is larger than s) if Q_i's block lies inside the region. Thus for large h's, the "root nodes" will represent maximal blocks of the image that approximately coincide with homogeneous regions.* The regions detected in this way must be "compact" in the sense that they can be approximated by square blocks. In Section 4 we will describe a top-down method of explicitly extracting from the image the set of pixels belonging to such a region.

More generally, we can use the polynomial fitting approach to detect coarse "features" such as spots, streak ends, streaks, and edges of all sizes in an image, by

*Many variations of this simple root node selection criterion can be formulated, based on different assumptions about the distribution of local property values in the image. Note that root nodes can be detected on many levels, and may represent nested pieces of the image; for example, in an image containing a cluster of small blobs, the individual blobs will give rise to root nodes at a low level of the pyramid, and the cluster will give rise to a root node at a higher level.

applying local operators at all levels of an intensity-summarized pyramid. For example, if a cell's block has large Fisher distances from all of its neighbors' blocks, the block must approximate an isolated "spot" in the image; and similarly for other feature types.

Features extracted in this way at any (sufficiently low) level of the pyramid, including the base itself, can serve as new "intensity arrays" that represent textural properties of the image. Intensity summarization and feature detection techniques can in turn be applied to these arrays to yield various texture-based analyses of the original image.

3. Bottom-up techniques: Contour-based

3.1. Summarizing contour data

In this section we describe methods of "summarizing" information about contours (edges or curves) in an image. We assume that these edges or curves are input to the base of the pyramid as a set of feature points, obtained by applying local edge or line detection operations to an intensity image (possibly itself the result of intensity summarization at some level of another pyramid). We refer to contours as "curves" from now on.

To detect smooth curves, each cell at level 1 examines its block of the image; if (many of) the feature points in the block lie on a smooth curve, the cell records the positions and slopes of the endpoints of the curve, and also stores pointers to the feature points that constitute the curve. Each cell at level 2 now examines the data provided by its children at level 1. If (many of) the endpoint data are consistent (i.e., they can be arranged in a sequence such that successive endpoints closely agree in position and slope),* the level-2 cell records the positions and slopes of the first and last endpoints, and stores pointers to the level-1 cells that contributed the consistent data. This process is repeated at levels 3, 4, $\cdots$. Thus if the image contains a long, smooth curve, some high-level cell will contain its endpoint data, and will also be the root of a tree of pointers whose leaves are the feature points comprising the curve.

*More generally, the "good continuation" of two curve ends can be measured by the total bending energy of the minimum-energy arc joining them.

This approach requires only a bounded amount of computation by the cells at each level, and so can be carried out in O(log image diameter) time, even if the curve is long. It has difficulties, however, if the image contains many long, smooth curves, or a single curve that doubles back on itself. In such cases, some of the cells will detect more than one smooth curve, and will need the capacity to store many sets of endpoint data. For a sufficiently "busy" image, this capacity will eventually be overloaded. At this stage, the cell can no longer store complete information about all the curves it sees; it can only store statistics of the properties of these curves, constituting a description of its image block as a "curve texture". (It can, in particular, store information about bimodality, anomalies, etc., as in Section 2.1.)

In general, for each stored curve (or for the ensemble of curves, if there are too many to be stored individually), the cell should record not only endpoint data, but also various global properties, such as average gray level (or the averages of the adjacent gray levels on each side, in the case of an edge), arc length, "wiggliness" (= total absolute curvature), etc. These properties can be estimated by combining property values obtained from the cells on the level below. It would also be useful to fit straight lines (or possibly higher-degree) polynomials to the curves; this too can be done recursively, based on the fits computed on the level below. As in Section 2.2, the fit error measure should be divided by the number of points being fitted; this reflects the intuitive fact that the fit to a long piece of wiggly straight line is "better" than the fit to a short piece. Some early experiments on curve summarization are described in [5].

A line fitting process can also be used to detect corners, or angles, on a curve [6]. A cell detects an angle if two of its children contain good straight line fits that have different slopes and that (approximately) meet at a common endpoint. The closer the fits, the smaller the slope difference need be for an angle to be detected.

3.2. Detecting "closed" contours (blobs or ribbons)

Summarized contours can be used to extract regions from an image that are "surrounded", on all sides or on two opposite sides, by edges or curves.

To detect "blobs", we begin by detecting edges (or curves) in the input image, and inputting the resulting set of feature points to the base of the pyramid. We then pass this contour information up through the pyramid, condensing and "summarizing" it at each stage, e.g., by straight line fitting. The cells at each level also examine their neighboring cells (e.g., in a 5-by-5 neighborhood) and check whether they are locally (nearly) "surrounded" by lines that run (approximately) broadside to

them (and in the case of edge data, that have consistent contrasts—i.e., their dark sides all face inward or all face outward). If a cell discovers that these conditions are satisfied, it has detected a "blob".

An early implementation of this blob detection scheme is described in [7]. A non-pyramid implementation, which detected blobs by searching out from each pixel in a set of directions, up to a distance equal to the maximum expected blob radius, is described in [8]. The pyramid approach has the advantage that it requires only local searches; blobs of any size will become locally detectable at some level of the pyramid.

The contour-based approach to blob detection may be more perceptually plausible than the intensity-based scheme described in Section 2.2. In the absence of edges, a region may look uniformly bright even when its brightness varies substantially from one side to another, due to shading; this suggests that the visual system cannot easily determine the slope of an (e.g.) linear fit to the brightness data. Conversely, suppose a region is surrounded by "pseudoedges" at which there is a local brightness change, but where the change smooths out as we move away from the edges in either direction, so that the regions inside and outside the edges actually have the same uniform brightness (except near the edges). In this situation the region on the dark side of the edges looks uniformly darker than the region on the bright side, even though the brightnesses of the regions are actually the same; this phenomenon is known as the Craik-O'Brien-Cornsweet illusion [9]. This suggests that the visual system assigns a brightness to a region based on the local brightnesses along the edges surrounding the region, rather than by fitting a function globally to all the gray levels in the region.

An edge- (or curve-) based pyramid technique can also be used to extract ribbon-like regions from an image. Here too we begin by inputting the feature (edge or curve) points to the base of the pyramid, and performing repeated (e.g.) straight line fitting at successively higher levels. The cells at each level also examine their neighborhoods (e.g., 5-by-5) and look for "antisymmetric" pairs of edges located on opposite sides of them—i.e., pairs of edges such that the given cell is on the dark side of both of them, or on the light side of both. In effect, this process detects "smoothed local symmetries" [10] at each level of the pyramid. When a cell detects such a configuration, it stores the estimated position and orientation of the local axis of symmetry, as well as the distance and angle between the pair of edges; the latter information defines a linear fit to the "width function" defined by the pair of edges. Cells on higher levels can then link pieces of symmetry axis into smooth curves along which the width function varies smoothly, by a straightforward generalization of the

methods described in Section 3.1. Global functions (such as straight lines) can also be fit to the axes and to the width function.

4. Top-down techniques: Feature delineation

The image regions detected using the intensity-based methods described in Section 2.2 can be extracted from the image by a top-down tree growing process such as the following: Take the root node P as the root of the tree. Let P' be a cell on some level $i < h$, and let $Q'_1, \ldots, Q'_k$ be the cells on level $i + 1$ whose image blocks overlap that of P'. Let Q' be that one of the Q's having smallest Fisher distance from P'. Then we make P' a tree node if and only if Q' is a tree node. This process is carried out for $i = h - 1, h - 2, \cdots$. The leaves of the tree at level 0 are the pixels constituting the region. There are several variations on this tree growing procedure; the details are described in [11]. An alternative formulation, in terms of down projection and boundary adjustment, is described in [12].

The region extracted from the image in this way is necessarily compact, since it cannot contain pixels that are far away from the original root node's block of the image. (Indeed, at successive stages of the tree growing process we can only add to the tree cells whose blocks overlap that of P, then cells whose blocks overlap these blocks, and so on, where the sizes of the blocks are exponentially decreasing.) Of course, we can also lose parts of P's block, since some blocks that overlap P's block will not meet the criterion for inclusion in the tree. Note that at the levels immediately below h, it is likely that the cells whose blocks are contained in P's block will nearly all join the tree, since they represent relatively large samples of the same image region; thus the union of these blocks will define a "solid" region. At still lower levels, the blocks may no longer resemble that of P, due to noise, but if they are near the center of P's block they will have no choice but to join the tree, so that the region will remain solid. Blocks near the edges of the region, however, will not be forced to join the tree, so the tree growing process will be able to closely approximate the shape of the region. It has been found that this approach is quite effective at extracting compact homogeneous regions from an image, even if they have irregular shapes.

A similar method of top-down "coloring in" can be used to extract a blob (or ribbon) detected by the methods described in Section 3.2 [7]. When we detect a pixel surrounded by edges, we give the pixel the average of the gray levels associated with the insides of the edges. We then go down to the next level of the pyramid (containing the pieces out of which the edges were summarized), close any (small!)

gaps in the edges, and extend the pixel's gray level to the higher-resolution pixels on the insides of the edges. (We can also optimize the positions of the edges by adjusting their positions to maximize the contrast between the inside and outside regions, as described in [12].) This process can be repeated until the bottom level is reached.

It is well known that, given a region defined by a subtree of the pyramid, the root node can compute many properties of the region in time proportional to the tree height, using divide-and-conquer techniques. Good approximations to some of these properties can even be obtained while the tree is still growing, before the bottom level has been reached. It would be of great interest to develop methods of allowing these property value estimates to influence the tree growing process; this would constitute an unprecedentedly close cooperation between the processes of image segmentation and region property measurement.

5. Extensions

In the preceding sections we have, for simplicity, treated the image data two-dimensionally, without reference to the fact that the underlying scene may be three-dimensional or that the image may be part of a time sequence. However, our approach can also be applied to infer three-dimensional scene information from an image, and to analyze optical flow in a sequence of images.

Global nonlinear variations in the gray level of a region may be an indication that the region is the image of a curved surface. Some work has been done using global function fitting to the image gray levels to detect specific types of surfaces in a scene [13]; this type of computation could be efficiently implemented using pyramid techniques such as those described in Section 2.2. Similarly, global variations (linear or nonlinear) in the sizes and spacings of texture elements over a region can serve as an indication that the region is the image of a slanted or curved surface. Thus the function fitting methods of Section 2.2 can serve as aids in the computation of surface shape from either shading, texture, or both.

Our function fitting methods can be applied to vector-valued as well as scalar valued data; for example, they can be applied to color data. If we are given a time sequence of images, and can estimate a set of displacement vectors relating points in one image to those in the next, we can also apply the function fitting approach to detect global patterns in the resulting (possibly sparse) "optical flow field". This approach has been successfully used to segment noisy synthetic optical flow fields into regions representing different types of motions, including translation, planar rotation, and scale change [14].

6. Concluding remarks

It should be pointed out that the pyramid techniques described in this paper are quite different from the ways in which pyramids have been used by other investigators (see, e.g., [15]). Pyramids are often used to generate a set of bandpass-filtered, sampled versions of an image. Our use of pyramids is quite different; we employ them for model *fitting* rather than for *filtering*.

We believe that the concepts outlined in this paper constitute a basic contribution to the methodology of vision systems design. Our approach makes use of the pyramid cellular architecture to rapidly compute global information about an image in a recursive fashion. The following are some of the key aspects of the approach:

- It provides a unified method of detecting various types of global patterns by bottom-up recursive fitting of low-order polynomial models to the data
- It provides a method of delineating the detected patterns by top-down recursive refinement of the fitted data
- It allows for the detection of more complex types of global patterns by applying local feature detection processes to the fitted models
- The methods can be applied to gray level surfaces, to edges or curves, or to vector-valued data such as disparity or optical flow fields

The transition from local to global—from pixel arrays to descriptive data structures—has traditionally been a major point of discontinuity in vision systems. Our approach offers the promise of making this discontinuity much less abrupt.

References

(1) D. Marr, *Vision*, Freeman, San Francisco, 1982, p. 95.

(2) M. Wertheimer, Principles of perceptual organization, in *Readings in Perception* (D.C. Beardslee and M. Wertheimer, eds.), Van Nostrand, Princeton, NJ, 1958, pp. 115–135.

(3) S.M. Dunn, L. Janos, and A. Rosenfeld, Bimean clustering, *Pattern Recognition Letters* **1**, 155–160, 1983.

(4) P. Burt, Hierarchically derived piecewise polynomial approximations to waveforms and images, TR-838, Computer Vision Laboratory, University of Maryland, November 1979.

(5) T.H. Hong, M.O. Shneier, R.L. Hartley, and A. Rosenfeld, Using pyramids to detect good continuation, *IEEE Trans. SMC*-**13**, 631–635, 1983.

(6) R.L. Hartley and A. Rosenfeld, Hierarchical line linking for corner detection, in *Integrated Technology for Parallel Image Processing* (S. Levialdi, ed.), Academic Press, New York, 1985, pp. 101–119.

(7) T.H. Hong and M. Shneier, Extracting compact objects using linked pyramids, *IEEE Trans. PAMI*-**6**, 229–237, 1984.

(8) L.G. Minor and J. Sklansky, The detection and segmentation of blobs in infrared images, *IEEE Trans. SMC*-**11**, 194–201, 1981.

(9) T.N. Cornsweet, *Visual Perception*, Academic Press, New York, 1970, p. 27

(10) J.M. Brady and H. Asada, Smoothed local symmetries and their implementation, AIM 757, Artificial Intelligence Laboratory, MIT, February 1984.

(11) A.D. Gross, Multiresolution object detection and delineation, TR-1613, Center for Automation Research, University of Maryland, January 1986.

(12) E.S. Baugher and A. Rosenfeld, Boundary localization in an image pyramid, TR-1488, Computer Vision Laboratory, University of Maryland, May 1985.

(13) R.M. Bolle and D.B. Cooper, Bayesian recognition of local 3-D shape by approximating image intensity functions with quadric polynomials, *IEEE Trans. PAMI*-**6**, 418–429, 1984.

(14) R.L. Hartley, Segmentation of optical flow fields by pyramid linking, *Pattern Recognition Letters* **3**, 253–262, 1985.

(15) A. Rosenfeld, ed., *Multiresolution Image Processing and Analysis*, Springer, Berlin, 1984.

SEGMENTATION OF TEXTURED IMAGES BY PYRAMID LINKING

Bradley P. Kjell

Computer and Information Science Department
George Mason University
Fairfax, VA 22030

Charles R. Dyer

Computer Science Department
University of Wisconsin
Madison, WI 53706

A pyramid linking algorithm for texture segmentation is presented. It is based on the computation of spatial properties of long, straight edge segments at fixed orientations. Features are computed for each edge segment in terms of the distances to the nearest neighboring edge segments of given orientations. This produces a set of sparse "edge separation maps" of features which are then used as the basis of a pyramid linking procedure for hierarchically grouping edges into homogeneously textured regions. Segmentation is performed in one bottom-up pass of linking nodes to their most similar parent. Results are shown using both the raw and smoothed edge separation features. All of the steps of the procedure can be efficiently implemented as parallel operations on a pyramid machine.

1. Introduction

Pyramid structures have been used in computer vision for many purposes (see [1] for a recent collection of papers). Primitive operations which have been used in many of these algorithms include (a) filtering/transform operations which construct multiresolution representations of images or other 2-D spatial data, (b) feature selection/detection operations which apply matching or interest operators, (c) projection operations which interpolate, fill, and expand a finer resolution description based on coarse resolution properties, (d) "homing" operations which are used for hypothesis verification and spatial focusing, and (e) "linking" operations which specify grouping rules for segmentation and region growing based on multiple scales of description. In each of these operations the pyramid is indexed in very well-structured ways: filtering operations are performed bottom-up through the pyramid, feature selection is performed at a single level, projection and homing operations are top-down processes, and linking operations usually make one or more alternating passes up and down the structure.

In this paper we describe a pyramid-based algorithm for texture segmentation. The basic operations used in the algorithm include all of the above except (d). Thus it represents another strong indicator of the importance of pyramid structures for vision and also supports the hypothesis that most basic pyramid processes are highly structured. Another important

NATO ASI Series, Vol. F 25
Pyramidal Systems for Computer Vision
Edited by V. Cantoni and S. Levialdi
© Springer-Verlag Berlin Heidelberg 1986

contribution of this paper is to show how pyramid algorithms can be effectively used with sparse iconic data. The method presented is based on the detection and grouping of long, straight edge segments in order to segment an image into regions containing different textures. Because edge segments are relatively sparse, yet a final partition of the image is required to define a complete segmentation, we needed to develop new techniques for both grouping sparse edge properties and interpreting the areas between edge tokens.

Our previous work showed that using spatial properties of edge segments as features for texture classification is very successful [2,3]. This paper shows that similar features can also be used successfully for texture segmentation. Section 2 describes the steps associated with the detection of the extended edge segments and the computation of inter-edge separation features. Section 3 presents the pyramid linking algorithm which groups similar features using the upper levels of the pyramid.

2. Edge Separation Features

In this section we describe a procedure for detecting long, straight edge segments and measuring inter-segment spatial separation features. Edge separation features will be used to describe a texture in terms of the distances between edge segments of various orientations. For example, each extended edge of orientation 0° will compute its average distance to the nearest extended edge of orientation $45i$°, for $i = 0, ..., 7$. We will detect extended edges in eight orientations, so a total of 64 possible texture features will be defined.

2.1 Extended Edge Detection

The goal of the edge detection step is to find long, straight edges of well-defined orientations. These extended edges are tokens more complex than pixels since they are built up in several steps from evidence collected over large areas of the image. Since the texture method will measure spatial relations between these edge-based tokens, for simplicity we will detect extended edge segments, each of which has a single global orientation. Hence in contrast to many edge detectors which attempt to follow contours around corners, our detector will find segments which are approximately straight and oriented in one of eight fixed orientations (multiples of 45°). Orientations range from 0° to 360°, hence the opposite sides of a square will have orientations 180° apart. Preliminary experiments determined that a greater number of orientations leads to little improvement in the method.

Extended edges are detected in several steps. For most steps, the processing for each orientation takes place independently of the other orientations. Typically a step takes an input map for a single orientation and transforms it to an output map for that orientation. Only in step 3, the weak-edge removal step, are maps from several orientations used together.

In any event, all steps can be performed very efficiently using the bottom 2-D array in a pyramid machine. The steps of the method are:

Step 1: Eight smoothed directional first derivative operators are applied to the image, one operator per orientation. The output of this step is eight edge strength maps, one for each orientation.

Step 2: Eight smoothed directional second derivative operators are applied to the image, one operator per orientation. Zero-crossings in the output of these operators indicate locations of edges. At each zero-crossing, the edge magnitude from the corresponding edge strength map is written to the output. The output of this step is eight zero-crossing maps.

Step 3: Weak edges are removed from the zero-crossing maps. That is, edge pixels in a zero-crossing map are suppressed if stronger edge pixels occur at approximately the same locations in another zero-crossing map. The output of this step is eight edge maps.

Step 4: Adjacent edge points in an edge map are "linked" together to form straight, extended edge segments. Small gaps are also filled and short edge segments are removed.

Step 5: The final step is to label the extended edges in each map. Each chain of pixels in an edge map is assigned a unique label. The label is given to each pixel of the chain. At this point the maps contain labeled, extended edges, and edge strength information has been discarded.

The final output of the edge detection stage is eight maps of extended edges, a map for each orientation. Each map contains edges of a single orientation and each extended edge is a connected chain of pixels with the same label. Notice that a single pixel may be part of several extended edges, each in a different orientation map. These eight maps are the only information used by the texture method described below. Hence grayscale intensity values are not used; we only require extended edge maps representing sets of edge segments which are thin and approximately straight.

2.2 Computation of Spatial Properties between Extended Edges

Following the detection of extended edge segments, we next compute a set of spatial properties of these edges. For simplicity we have investigated statistics of *closest pairs of extended edge segments.* That is, for each extended edge segment we compute the value d_i which is the average distance from the given edge to the nearest edge of orientation $45i°$. Hence associated with each edge is a vector of eight spatial feature values. Previous studies [2,3] showed that these features are very effective for texture classification.

More specifically, given a set of extended edges, the following procedure is used to compute the set of eight *edge separation maps* (one for the nearest extended edge at each orientation):

Step 1: Compute the generalized digital Euclidean Voronoi diagram for each extended edge map [4]. This can be done in parallel at the bottom level of the pyramid in $O(n)$ time for an n by n image. The generalized discrete Voronoi diagram divides the digital plane into disjoint regions. Each region contains those points that are closest to a particular extended edge. The output of this step is an image in which each pixel is labeled with a triple, (dx, dy, λ), giving the displacement, (dx, dy), to the nearest extended edge and the label, λ, of that edge.

Step 2: For each pair of directions *dir* 1 and *dir* 2:

(a) Superimpose the edge map for *dir* 1 on the Voronoi diagram for *dir* 2 edges. Each point in the Voronoi diagram contains the displacement (dx, dy) to the nearest *dir* 2 edge.

(b) For each *dir* 1 edge, compute its average distance to the closest *dir* 2 edge by averaging $(dx^2 + dy^2)^{1/2}$ for all pixels contained in the edge. Relabel each pixel contained in the edge with this separation distance.

(c) Set all non-edge pixels to have value UNDEFINED.

The separation between edges of the same orientation can be calculated by examining just the Voronoi diagram for that orientation. The result for each pair of edge orientations is a set of labeled extended edges, where each edge of orientation *dir* 1 is labeled with its average distance to the nearest edge of orientation *dir* 2. All the pixels of a particular edge will have the same label (distance), and, of course, several different edges can have the same label. (It is desirable that edges from the same texture all have approximately the same label since this corresponds to the fact that the spatial relations between pairs of edges are approximately uniform over the entire texture.) This procedure computes eight edge separation maps resulting in a description of the distance from each extended edge to the closest extended edge of a given orientation. In practice, however, Step 3 can be carried out for only a few selected pairs of directions, corresponding to *a priori* knowledge about the textures' principal directionalities.

Fig. 1 shows an image containing two textures, raffia above the 45° diagonal, and pigskin below. Fig. 2 shows a Straw/Water texture image. These textures have been histogram flattened to 64 gray levels; the distribution of gray levels is the same for each texture. The 0° extended edges for the Raffia/Pigskin image are shown in Fig. 3. Notice that the separations between edges in the pigskin texture are visually different "at a glance" from

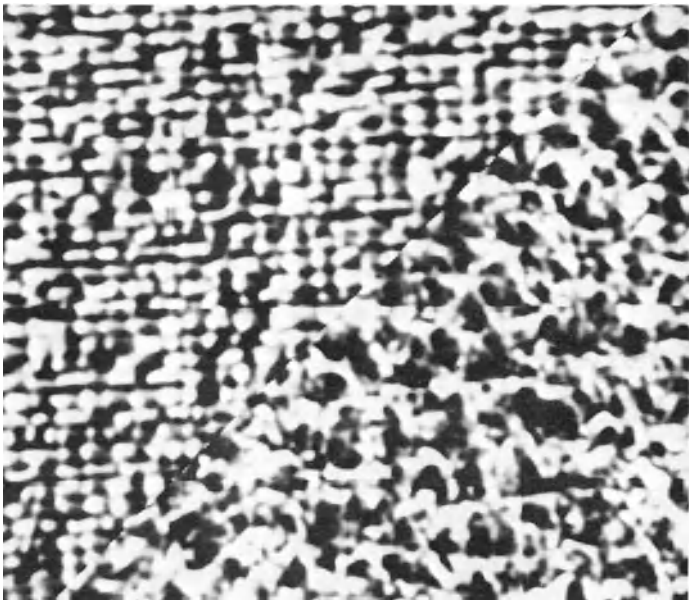


Fig. 1. Raffia/Pigskin texture image.

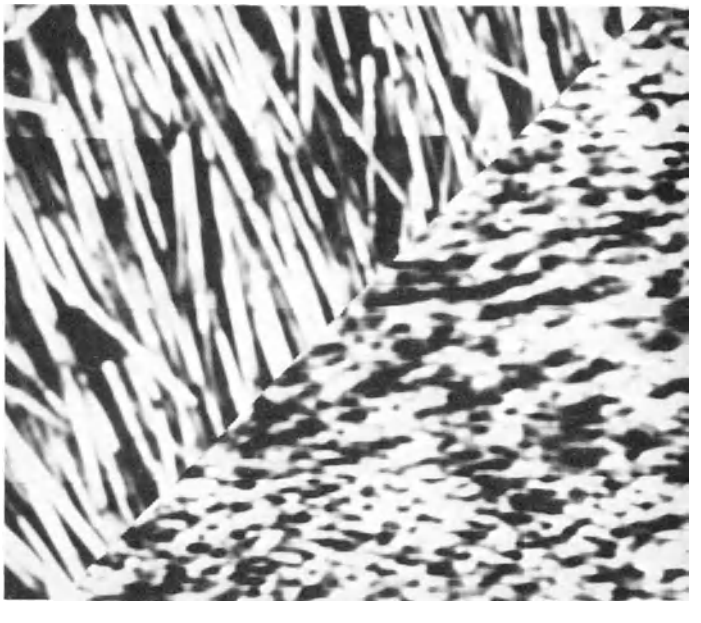


Fig. 2. Straw/Water texture image.

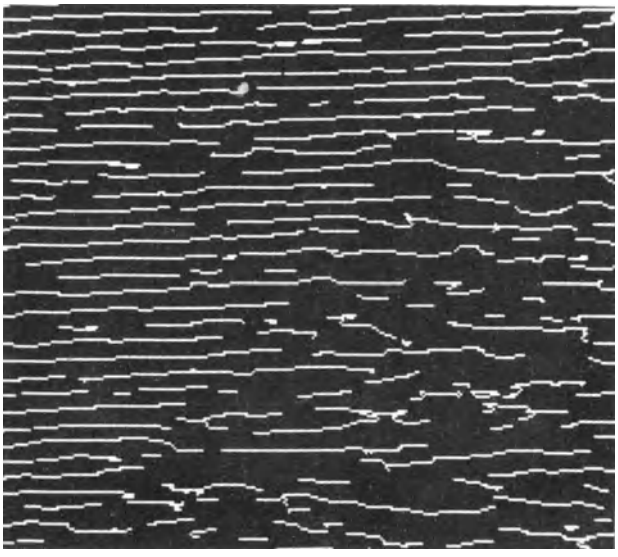


Fig. 3. Extended edges in direction 0° for Raffia/Pigskin image.

the separations between edges in the raffia texture.

3. Segmentation using Edge Separation Features

Most previous methods for texture segmentation have used pixel grayscale properties [5,6] or features based on the gray levels in a small block centered at each pixel [7]. Our method is based on higher level symbolic tokens and hence does not compute a feature for each pixel. Our goal is to group extended edges into distinct textural regions by merging areas of the image into regions containing similar extended edge features. Each region will be represented by an "average" edge separation vector over all of the extended edges contained in the region. There are two major difficulties with this type of approach. First, regions must be carefully grown so that extended edges from more than one texture are not included in the same region. Second, the method must be computationally feasible — that is, exponential time solutions which consider a combinatorially explosive number of possible solutions are prohibitively expensive.

In this section we define a class of methods that (a) segments images to pixel level precision using extended edge segment features instead of pixel properties, (b) uses spatial context of regions of uniform texture, (c) does not use training data, (d) computes texture properties at multiple levels of resolution, and (e) iteratively merges regions in parallel so that regions can grow in size at an exponential rate. The algorithms described below to achieve all these goals will use an overlapped pyramid structure [8] for the purposes of (a)

hierarchically organizing the computation of more and more global textural features (multi-resolution feature computation), and (b) imposing a regular control strategy for hierarchical region growing (hierarchical grouping rules). The overlapped pyramid is defined so that each node has 16 children in a 4 by 4 block at the level below, and 4 parents in a 2 by 2 block at the level above. Blocks overlap by 50 percent both horizontally and vertically so the height of the pyramid is the logarithm of the size of the base.

3.1 Grouping Extended Edge Features

Fig. 3 suggests that the extended edge separation maps can contain sufficient information for the purposes of texture segmentation. Preliminary experiments determined that individual extended edge segments cannot be reliably classified, however. Although the average edge separations for a single texture region can reliably be used in classification, the properties of individual edges vary too greatly. "Pyramid linking" algorithms provide a more robust class of grouping methods. These algorithms are based on parallel, iterative growing of regions in which the pyramid structure is used to constrain the number and types of regions which are considered for merging, and to impose a hierarchical control strategy on the process.

In particular, we have investigated several methods based on the bottom-up, iterative linking algorithm in [9]. This algorithm constrains how regions can be merged by the implicit links in an overlapped pyramid in which each node is connected to four parent nodes at the next higher level. The algorithm iterates at a given level until all nodes are linked to their "best" parent node at the next level. This process is then repeated at the next higher level, and so on until one pass up the pyramid is completed. These links thus provide a mechanism for merging regions (represented by nodes at a given level) into larger regions (corresponding to the parent nodes at the next higher level). In the algorithms used here, nodes are always forced to choose a best parent. This insures that there will be at most four regions represented at the level below the apex. We then group these regions into the most similar pair of regions. Each node stores a vector of property values corresponding to the set of edge separation maps used. The algorithm, called SEGMENT, is given as follows:

```
procedure SEGMENT (Image)
begin
  Pyramid := INIT_LEVEL_0(Image);
  LINK(Pyramid);
  LABEL(Pyramid);
end;
```

```
procedure LINK (Pyramid)
for k := 1 to Top - 1 do
  begin
    foreach node j at level k do
      INITIALIZE_PROPERTY_VECTOR(j);
    do
      begin
        foreach node i at level k-1 do
          UPDATE_PARENT_POINTER(i);
        foreach node j at level k do
          UPDATE_PROPERTY_VECTOR(j);
      end;
    until No_links_updated;
  end;
```

INIT_LEVEL_0 reads one or more edge separation maps and assigns to each level 0 node a vector of values from these maps. For the most part we have used just a single edge separation map. Note that most nodes will have an UNDEFINED value since only pixels which are part of an extended edge will have a defined value in an edge separation map. If several edge separation maps are used, the vector for a node may have one, several, or all values UNDEFINED. LABEL assigns each region at level Top-1 a unique label and then projects this label to all of its descendant nodes, in particular labeling all level 0 pixels with the final region label.

UPDATE_PARENT_POINTER(i) picks the parent node with most similar feature vector to child i's. If a single property value is used, the child chooses the parent closest in value to its own. If the child's value is not defined, it picks arbitrarily any parent with a defined value. If no parent has a defined value, the choice is also arbitrary. If several property values are used, the closest parent is the one with the least average difference between defined property values. UPDATE_PROPERTY_VECTOR(j) computes the average of each feature over all the children nodes currently linked to parent j. UNDEFINED values are not used in computing the average; if no children have a defined value for a particular feature then that feature is UNDEFINED in the parent.

The result of applying SEGMENT to the Straw/Water image using the edge separation map for 0° edges to 180° edges is shown in Fig. 4. The segmentation for this image is 81% correct. The results of the algorithm for various edge separation maps using the Raffia/Pigskin image are summarized in column b in Table 1. Results for the Straw/Water image are given in column b in Table 2.

3.2. Grouping Smoothed Extended Edge Features

The results in Tables 1 and 2 indicate that using the raw edge separation map data as input to SEGMENT are unreliable at the lowest levels of linking where regions are merged

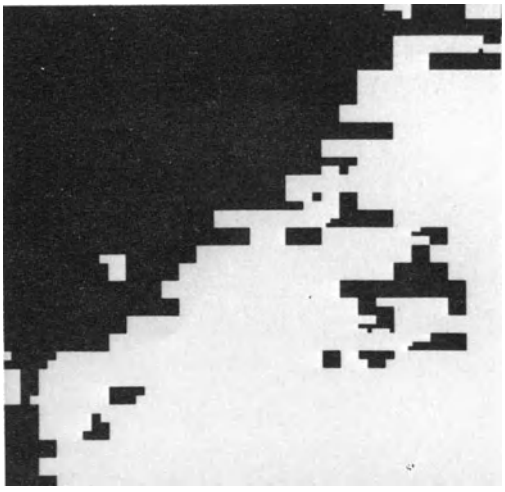


Fig. 4. Segmentation of Straw/Water image using 0-180 edge separation map.

Edge orientations	Edge sep. map	Smoothed separation map level 2	level 3	level 4	Optimal threshold level 2	level 3	level 4
0-180	77.50	88.19	91.25	91.78	78.14	84.71	93.99
0-270	66.80	75.32	71.04	65.63	59.21	60.42	61.31
45-225	73.20	75.77	72.80	76.83	61.18	61.78	64.76
45-315	76.57	59.11	63.83	76.08	76.72	82.11	88.67
90-0	86.28	91.08	92.79	95.99	86.26	91.65	95.38
90-270	69.34	61.33	64.77	68.21	59.03	61.81	64.96
135-45	71.60	66.72	62.27	73.68	57.41	53.02	53.62
135-315	65.17	46.94	67.53	65.02	71.70	77.33	86.73
180-0	79.75	85.57	89.86	91.18	80.25	85.47	91.97
180-90	59.15	78.77	75.91	70.37	67.57	70.43	75.78
225-45	61.42	62.58	73.85	74.31	63.61	64.82	68.19
225-135	67.10	82.07	87.31	76.96	75.35	78.77	87.23
270-90	59.60	56.87	63.10	67.97	60.13	60.85	61.65
270-180	85.62	92.79	96.64	93.63	82.90	89.27	94.19
315-135	79.23	66.99	75.77	88.25	73.29	76.28	84.29
315-225	67.92	72.94	62.33	61.00	59.72	57.14	55.93
4 anti	75.87	89.26	76.52	85.21	81.66	84.84	90.58
4 best	92.14	93.80	94.93	91.51	90.61	94.01	97.55
4 worst	75.29	69.95	82.68	75.29	67.58	71.65	69.60
all 16	60.99	72.83	76.98	93.85	91.06	95.08	95.94
(a)	(b)	(c)	(d)	(e)	(f)	(g)	(h)

Table 1. Percentages of pixels segmented correctly for Raffia/Pigskin image. (a) Orientation of the edges used. (b) SEGMENT used with raw edge separation map. (c)-(e) SEGMENT used with BUILD and PROJECT from level 2, 3, and 4, respectively. (f)-(h) Optimal thresholding using smoothed separation features projected from level 2, 3, and 4, respectively.

based on very small windows of the image. That is, the initial levels of bottom-up linking are error prone because most of the image area being evaluated by a node contains no edges (UNDEFINED children) and the edges which are present may be locally very variable in their feature values. Some top-down information is clearly needed to supplement this strictly bottom-up procedure. To solve this problem we have defined two additional procedures, BUILD and PROJECT (based on [10]), in order to first smooth extended edge property values over small neighborhoods in order to reduce these two effects. Specifically, these algorithms are defined as follows:

```
procedure BUILD (pyramid);
for L := 1 to Top do
   foreach node at level L do
      if at least one child has a defined value then
         node := weighted average of children's defined values;
      else
         node := UNDEFINED;

procedure PROJECT (pyramid, start);
for L := start - 1 to 0 do
   foreach node at level L do
      if at least one parent has a defined value then
         node := weighted average of parents' defined values;
      else
         node := UNDEFINED;
```

Notice that in procedure BUILD only the nodes with defined values at level 0 contribute to the values of nodes at level 1. Each level 1 node inspects a 4 by 4 block of children for defined values. Any defined value in this block will contribute to the parent's value. Level 1 will thus have proportionally more defined values than level 0, and so on up the pyramid. Thus at any level, a node's value is an average of the defined values of its descendants at level 0. In an overlapped pyramid, a node at level k projects down to a block with $3 \cdot 2^k - 2$ pixels on a side. Thus most nodes at the highest levels will have defined values.

PROJECT starts from a predefined level and projects the values at this level down to each node's 16 children, and so on down to level 0. The starting level is selected so that most nodes will have defined values, and thus most nodes at level 0 will have a defined value.

PROJECT applied to a pyramid constructed from an edge separation map using BUILD serves two purposes: (a) most pixels at level 0 which are not part of an extended edge will have their UNDEFINED values replaced by values representing the smoothed averages of their nearest extended edges, and (b) pixels at level 0 which are part of an extended edge will have their separation values smoothed using the separation values of other nearby edge points. The degree of smoothing is determined by the starting level used in PROJECT. Fig. 5 shows the result of this smoothing process for the Raffia/Pigskin image. 0° edges were

Edge orientations	Edge sep. map	Smoothed separation map level 2	level 3	level 4	Optimal threshold level 2	level 3	level 4
0-180	80.96	87.93	88.71	97.11	88.92	93.47	95.70
0-270	85.69	88.77	91.74	79.08	71.85	78.87	83.28
45-225	86.05	90.46	94.34	91.85	82.75	88.32	86.67
45-315	51.15	62.65	64.47	68.56	62.54	63.18	65.11
90-0	79.21	90.64	85.51	91.57	85.39	90.39	93.45
90-270	74.45	84.35	84.25	86.02	71.60	76.03	84.07
135-45	78.17	68.31	71.11	83.72	70.52	74.07	79.80
135-315	71.90	65.87	67.64	80.37	59.05	60.05	66.17
180-0	85.33	81.09	92.69	86.62	88.27	92.02	96.36
180-90	59.03	74.86	70.13	70.15	63.55	71.75	81.77
225-45	69.46	81.68	77.84	91.61	83.18	81.66	84.77
225-135	58.06	68.15	64.61	58.96	69.86	58.94	63.25
270-90	70.10	83.19	83.11	82.83	69.50	70.77	78.45
270-180	89.63	91.33	91.85	94.21	87.35	94.40	96.87
315-135	69.34	80.32	77.38	77.69	56.58	54.81	53.54
315-225	83.36	86.17	85.03	83.21	80.55	86.43	87.18
4 anti	91.95	92.63	94.27	95.94	89.98	94.65	96.76
4 best	90.85	92.46	91.54	96.50	95.34	96.87	97.55
4 worst	71.25	68.17	74.54	75.11	65.67	65.74	69.70
all 16	94.70	95.24	95.65	95.57	93.96	94.71	97.51
(a)	(b)	(c)	(d)	(e)	(f)	(g)	(h)

Table 2. Percentages of pixels segmented correctly for Straw/Water image. (a) Orientation of the edges used. (b) SEGMENT used with raw edge separation map. (c)-(e) SEGMENT used with BUILD and PROJECT from level 2, 3, and 4, respectively. (f)-(h) Optimal thresholding using smoothed separation features projected from level 2, 3, and 4, respectively.

initially labeled with their average distance to the nearest 180° edges. These values were then smoothed by applying BUILD and PROJECT from level 4.

The smoothed separation maps constructed by the above procedures are now used as input to level 0 for SEGMENT. Fig. 6 shows the resulting segmentation of the Straw/Water image using the 0° to 180° edge separations and projection from level 4. Columns c, d, and e in Table 1 show the segmentation accuracy with the Raffia/Pigskin image for projection from levels 2, 3, and 4 and for various edge orientation maps. Results are given in Table 2 for the Straw/Water image. Generally, in these examples, accuracy increases as the projection level increases. Level 5 was not used because the amount of smoothing, equivalent to using a 94 by 94 smoothing mask, was too large. Level 1 was not used because too many pixels are still UNDEFINED at level 0.

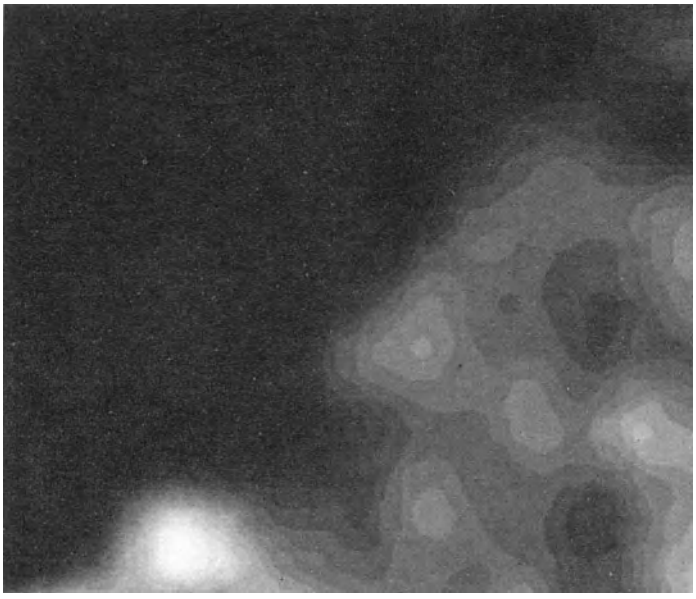


Fig. 5. Smoothed separation map for the Raffia/Pigskin image using the 0-180 edge separation map and smoothing using BUILD and PROJECT from level 4.

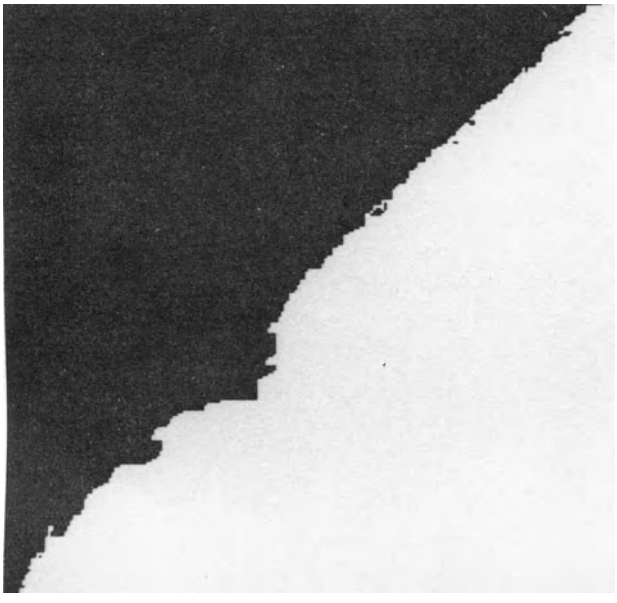


Fig. 6. Segmentation of the Straw/Water image using the smoothed 0-180 edge separation map.

As a final example, Fig. 7 is a close-up image of a raccoon lying in dried grass; the upper-left part of the image is the raccoon's fur and the lower-right part is the grass. The range of gray levels in this image is about the same for the raccoon fur and the grass. Consequently, segmentation by gray level thresholding would not work very well. The fur of the raccoon and the blades of the grass are both linear textures, but with different orientations. The orientation of the fur is mostly left-leaning vertical while the grass orientation is primarily horizontal. Based on this observation, two oriented edge separation features were chosen: the 0-180 separation between antiparallel horizontal edges, and the 135-315 separation between antiparallel, left-leaning vertical edges. Smoothed separation maps for these features were first constructed from the edge separation maps by projection from level 3. The resulting segmentation is shown in Fig. 8.

3.3 Comparison with Optimal Thresholding

In this section we compare the performance of procedure SEGMENT with optimal thresholding of the smoothed separation maps. To compute the optimal threshold, a histogram of a texture feature was computed for each region in an image, and the threshold was then selected by hand as the value which minimized the misclassification error. Note that this is the best threshold that can be found for the given feature because it is determined using knowledge of the ideal segmentation. Columns f, g, and h in Tables 1 and 2 show the segmentation accuracies using the optimal threshold with the smoothed separation maps. In general, SEGMENT performs better.

The experiments presented here all used a single edge separation map feature as the input to the segmentation algorithm. For example, the first row of Table 1 uses the separation from 0° edges to 180° edges. In general, it is often difficult to know *a priori* which particular edge separation map will be the best one. Preferably, an image should be reliably segmented without knowing in advance which feature(s) to use. To test how robust our algorithm is when several features of unknown utility are combined, SEGMENT was run using several sets of four features for the Raffia/Pigskin image. The lower part of Table 1 gives the percentage correct segmentation for these sets. Each of the four edge separation maps in the first set of features was made for antiparallel edges, specifically 0° to 180°, 45° to 225°, 90° to 270°, and 135° to 315°. The best of these features used individually was 78% correct, the others were somewhat worse. When all four were used the accuracy decreases only slightly to 76%. The next set used the four edge separation maps that performed best as individual features. The accuracy using all four was 92%, about a 6% increase over the best individual result. Using four smoothed separation maps built from the four antiparallel edge separation maps resulted in 77% correct segmentation. Similar results for the Straw/Water texture image are shown at the bottom of Table 2.



Fig. 7. Raccoon lying in grass.

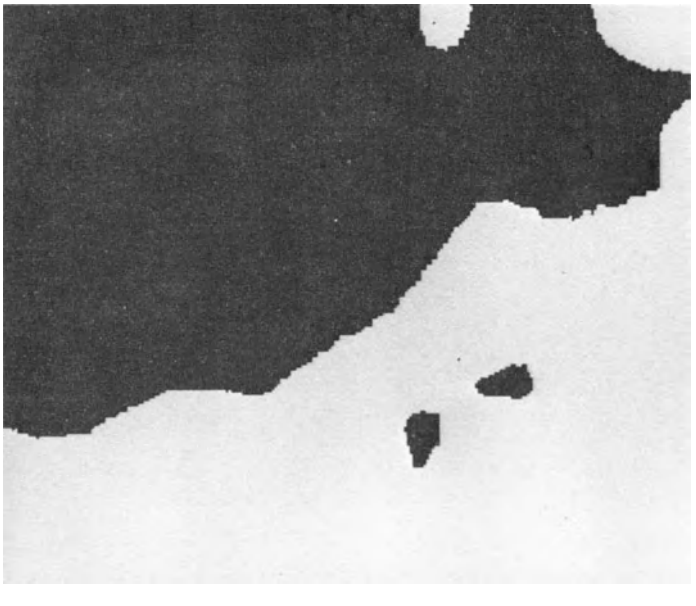


Fig. 8. Segmentation of the Raccoon/Grass image using two edge separation features.

The optimal threshold program also was modified so that it used several features. The optimal threshold was first calculated for each feature individually, and then each pixel was classified by having each threshold "vote" for its preferred region class. Ties were broken by including the votes from neighboring pixels. Using the smoothed separation maps for the four antiparallel edge pairs, optimal thresholding resulted in 91% accuracy while SEGMENT performed slightly worse. Using the best four smoothed separation maps, SEGMENT and optimal thresholding performed comparably, although results varied somewhat depending on the amount of smoothing (determined by PROJECT's starting level).

4. Concluding Remarks

The major contributions of this paper have been to show that:

- Features based on spatial properties of long, straight, extended edge segments are reliable features for texture description and segmentation.

  We defined inter-edge separation features in which each extended edge segment is described by its distance to its nearest neighboring extended edges at each of eight fixed orientations. Extended edges of this type are important tokens for general low-level vision, and this measure of the spatial context of an edge seems to capture key information needed for texture segmentation. Also, since an extended edge is unlikely to cross a boundary between two different textured regions, the "window size selection problem" has been dealt with as well. The neighbors of an extended edge, which its features are based on, are also likely to be in the same region as the edge. A final advantage of edge separation features is that an estimate of the best features for a given set of textures can often be made by visual inspection. Features involving the orientations that are dominant in the textures are likely to be the best for segmentation.

- Pyramid-based linking rules for hierarchically grouping sparse, but smoothed, extended edge segments into homogeneous texture regions are very effective and can be efficiently implemented in a pyramid machine.

  We showed that the raw edge separation features were too sparse and noisy to be used directly for texture segmentation. Rarely was the segmentation accuracy above 85% when using a single feature. By first smoothing the edge separation features, however, (using the bottom levels of the pyramid with procedures BUILD and PROJECT) these two problems were minimized and the results were significantly improved. Individual smoothed edge separation features resulted in up to 97% correct segmentations. In general, segmentation accuracy tended to increase with the size of the neighborhood used for smoothing (although some features are better with smaller neighborhoods). This suggests that a larger neighborhood than that defined by an edge and its nearest

neighbors would be beneficial. That is, the spatial context of an edge should be expanded to include more than just its closest neighbors.

Finally, it should be emphasized that all of the operations used in the procedures presented here are very efficiently implemented on a pyramid machine. The initial steps detect extended edges and then compute the separations between closest pairs of edges. This phase of token detection and feature measurement can be performed by the bottom level of the pyramid. The segmentation phase is then accomplished by smoothing (using BUILD and PROJECT to make one pass up to level 4 and then back down to level 0), linking (using LINK to make one pass up the pyramid), and final label projection (using LABEL to make one pass down the pyramid).

References

[1] A. Rosenfeld, Ed., *Multiresolution Image Processing and Analysis.* Berlin, West Germany: Springer-Verlag, 1984.

[2] B. P. Kjell and C. R. Dyer, "Edge separation and orientation texture measures," *Proc. IEEE Conf. Computer Vision and Pattern Recognition,* San Francisco, CA, June 1985, pp. 306-311.

[3] B. P. Kjell, "Edge separation and orientation texture measures," Ph.D. dissertation, Computer Science Department, University of Wisconsin, Madison, WI, 1985.

[4] T. Phillips and T. Matsuyama, "The labeled discrete Voronoi diagram," Technical Report TR-1278, Center for Automation Research, University of Maryland, College Park, MD, May 1983.

[5] P. C. Chen and T. Pavlidis, "Segmentation by texture using a co-occurrence matrix and a split-and-merge algorithm," *Computer Graphics and Image Processing,* vol. 10, pp. 172-182, 1979.

[6] M. Pietikainen and A. Rosenfeld, "Image segmentation by texture using pyramid node linking," *IEEE Trans. Syst., Man, Cybern.,* vol. SMC-11, pp. 822-825, 1981.

[7] K. I. Laws, "Texture energy measures," *Proc. Image Understanding Workshop,* Los Angeles, CA, Nov. 1979, pp. 47-51.

[8] P. J. Burt, T. H. Hong, and A. Rosenfeld, "Segmentation and estimation of image region properties through cooperative hierarchical computation," *IEEE Trans. Syst., Man, Cybern.,* vol. SMC-11, pp. 802-809, 1981.

[9] J. Cibulskis and C. R. Dyer, "Node linking strategies in pyramids for image segmentation," *IEEE Trans. Syst., Man, Cybern.,* vol. SMC-14, pp. 424-436, 1984.

[10] P. J. Burt, "The pyramid as a structure for efficient computation," in *Multiresolution Image Processing and Analysis,* A. Rosenfeld, Ed. Berlin, West Germany: Springer-Verlag, 1984, pp. 6-35.

Local motion estimation with the Dynamic Pyramid

Joachim Dengler
Department of Medical and Biological Informatics
German Cancer Research Center, D-6900 Heidelberg

Abstract

The Dynamic Pyramid is a tool for analyzing monocular moving image sequences. Its process for achieving correspondence between successive frames is based on the physical model of the elastic membrane, where the cost function consisting of the similarity between the frames and the deformation of the transformation vector field is minimized.
A new concept is developped in order to determine the iniatial local displacements. It includes isolated points like corners but also makes use of the one given component of the displacement at edges. For the local correlation the sign of the Laplacian is used. This is a fast and reliable method to determine the initial displacements. A new approach of region oriented continuity constraints is developped in order to allow semantic discontinuities e.g. due to occlusion.

To make a scaleinvariant analysis of image sequences possible, the membrane model is extended to the Dynamic Pyramid. It makes use of the fact that at the coarser levels of the pyramid even large distances come into the realm of local operators. The results of one level are good starting values of the next. Thus also there local operators can be applied.

This achieves a reliable mapping in combination with an efficient calculation. The result of the dynamic pyramid is a local motion vector field. From there segmentation of moving objects is simple and the basic information for further steps like 3D-reconstruction is given.

Introduction

This paper presents an approach to solve the correspondence problem in moving realworld scenes. The determination of the displacement vector field is local so that independently moving objects can be dealt with. The approach is completely bottom up, and no scene specific apriori knowledge is used.

The described approach is related to other methods based on optical flow (Horn und Schunck 1981, Prager und Arbib 1983, Hildreth 1984, Nagel und Enkelmann 1984), in fact it can be shown (Dengler 1985) that those algorithms are special cases of the general framework discribed here.

It makes use of some very general physical constraints of realworld scenes:

- individual objects are more or less rigid

- the displacement of objects between two successive frames is so small, that the 2D projection of the velocity-field is homogeneous within an object

NATO ASI Series, Vol. F 25
Pyramidal Systems for Computer Vision
Edited by V. Cantoni and S. Levialdi
© Springer-Verlag Berlin Heidelberg 1986

- grayvalue changes are potential object edges, therefore they are candicates for discontinuities of the displacement vector field

The method exploits only locally available information:

- isolated significant points give both components of the local displacement

- edges or lines give one component of the local displacement (aperture problem)

- areas of uniform grayvalue or texture are interpolated from locations, where information is available.

These aspects are all incorporated into the concept of the Dynamic Pyramid, at the core of which there is the method of elastic matching based on the physical model of the elastic membrane.

Elastic Matching

The model of the elastic matching is that of the 2-dimensional elastic membrane. It was first introduced at the University of Pensylvania for matching CT-slices (Broit 1981, Bajcsy et al. 1983). It has here been modified considerably to make it appropriate for motion detection. The aim of calculating the displacement vector field U is achieved by minimizing a global cost-function consisting of the local deformation of the vector field and the similarities of local image features:

cost = deformation energy - similarity energy

One can imagine one of the matching candidates to be drawn onto a rubber sheet, and matching is achieved by pulling the sheet until it fits best its partner image.

To achieve minimal cost, the local derivatives of the cost function are set zero, resulting in an equation of forces.

deformation force + "external" force = 0

The external forces are assumed to be derived from a potential V.

Assuming a homogeneous medium, this is expressed in the linear partial differential equation of the elastic membrane (Feynman 1964):

$$C_1 \Delta U + C_2 \nabla(\nabla \cdot U) - \nabla V = 0$$

C_1 and C_2 are the elastic constants related to the Lamé-constants:

$$C_1 = \mu \qquad C_2 = \lambda + \mu$$

The resulting displacement vector field is completely determined by the Potential $V(x,y)$ and the elastic constants C_1 and C_2. By introducing special types of Laplacians knowledge about discontinuities of the vector field can be included.

Determination of the external forces from local correlations

In the case of a real membrane the external forces are at the boundary or they are global. Here at every point forces are possible, and the potential function $V(x,y)$ can be different at each point $X = (x,y)^T$

$$F_{ij}(U) = F(X-X_{ij}) = - \nabla V(X-X_{ij})$$

The potential $V(X-X_{ij})$ is determined by local correlations.
It has been shown (Nishihara 1984) that the binary correlation of the sign of the Laplacian is an especially appropriate measure regarding resolution, controllable range, confidence as well as fast calculation. Changes of intensities are reflected in changes of the sign of the Laplacian. There is no necessity to determine explicit features like corners, edges etc. as long as the features extracted from both partner images reflect similar properties.

The cross-correlation signal C between the sign matrices S_1 and S_2 is

$$C_{12}(X-X_{ij}) = \frac{1}{|G|} \cdot \sum_{X' \in G} w(X') \cdot S_1(X+X') \cdot S_2(X_{ij}+X')$$

As G is a 2-dimensional area, X, X', X_{ij} are vectors. The weight-mask w is chosen to be of Gaussian shape, thus a smooth cutoff is achieved.

The correlation signal is approximated with a polynomial of degree 2. As the ideal approximation of the sign correlation signal is a cone, the best fit is achieved by the following model (Nishihara 1984):

$$V(X-X_{ij}) = (C(X-X_{ij}) - 1)^2$$

$$V(X-X_{ij}) = V_0 + (V_x \; V_y) \cdot (X-X_{ij}) + \frac{1}{2} (X-X_{ij})^T \begin{pmatrix} V_{xx} & V_{xy} \\ V_{xy} & V_{yy} \end{pmatrix} (X-X_{ij})$$

The curvature parameters V_{xx}, V_{xy}, V_{yy} are best determined from the autocorrelation signal, where V_x and V_y are ideally zero. It is assumed that any consecutive frames are similar enough so that the curvatures of the cross-correlation signal is well represented by those of the auto-correlation signal. The distribution of the principal curvatures indicates the quality and the range of the operator, from which the correlation signal is derived.

From the cross-correlation signal V_{12} the gradient components V_x and V_y are determined. To compensate asymmetries of the correlation signal, the gradient of the auto-correlation signal - which should be ideally zero - is subtracted:

$$(V_x, V_y) = \nabla V_{12}(0,0) - \nabla V_{22}(0,0)$$

So the initial forces are

$$F(U) = - \nabla V(U)$$

$$= - \begin{pmatrix} V_x \\ V_y \end{pmatrix} - \begin{pmatrix} V_{xx} & V_{xy} \\ V_{xy} & V_{yy} \end{pmatrix} \cdot U = - \begin{pmatrix} V_{xx} & V_{xy} \\ V_{xy} & V_{yy} \end{pmatrix} \cdot (U-U_0)$$

The Aperture problem and the Moore-Penrose Inverse

In order to find the initial displacements U_0, i.e. the point where the external forces vanish, the equation is inverted formally:

With $H = \begin{pmatrix} V_{xx} & V_{xy} \\ V_{xy} & V_{yy} \end{pmatrix}$

$$U_0 = - H^{-1} \cdot (V_x, V_y)^T$$

This Hessian Matrix H can be singular at flat regions or at edges. The latter case is equivalent to the aperture problem which says that at an edge only one component of the displacement vector can be determined locally. Both cases are dealt with appropriately by applying the Moore-Penrose Inverse instead of trying to invert the matrix with heuristic assumptions. Let

$$H = P\Theta^{1/2}Q^T$$

be the singular value decomposition of H, with P,Q being orthogonal and Θ the diagonal matrix of the eigenvalues of HH^T, then

$$H^* = Q\Theta^{-1/2}P^T$$

is the Moore-Penrose-Inverse of H.

In case $|\Theta_{ii}| < \varepsilon$ then $\Theta^{-1}{}_{ii} = 0$. If both eigenvalues are larger than the given threshold ε, then the Moore-Penrose-Inverse is idential with the usual Inverse. In the case of an edge the correct component orthogonal to the edge is found.

In order to determine the local forces reliably, V_x and V_y are projected onto the image of H, a procedure based on the Moore-Penrose-Inverse, where the components of $(V_x\ V_y)$ pointing to directions associated with vanishing eigenvalues of H are set zero.

The constants of the membrane equation

The constants C_1 and C_2 determine the deformation of the membrane with a given field of forces F.

If C_1, $C_2 << F$, a plastic deformation is achieved. The vector field U is determined only by the external forces $F = -\nabla V$. If C_1, $C_2 >> F$, only rigid transformations like translations or rotations are possible. This is equivalent to a global correlation. C_1 accounts for isotropical variations, whereas C_2 also includes shearing. This is not relevant in this context. Therefore one can put $C_2 = 0$. Thus the simplified equation is

$$C_1\Delta U - \nabla V = 0$$

In the discrete formulation, which can be found by the method of finite elements this results in

$$C_1 \cdot \begin{pmatrix} & 1 & \\ 1 & -4 & 1 \\ & 1 & \end{pmatrix} * u - V_{xx} \cdot u - V_{xy} \cdot v = V_x$$

$$C_1 \cdot \begin{pmatrix} & 1 & \\ 1 & -4 & 1 \\ & 1 & \end{pmatrix} * v - V_{xy} \cdot u - V_{yy} \cdot v = V_y$$

Using U_0 as initial values, this system of linear equations is solved by an iterative algorithm, from which the conjugate gradient method is the most reliable and the one to converge best, besides beeing conceptually parallel.

The regionoriented smoothness constraints

Most methods to determine the displacement vector field assume isotropical continuity, e.g. Horn and Schunck (1981) or Prager and Arbib (1983). Also Broit (1981) and Bajcsy (Bajcsy et al. 1983) consider only the case of isotropical continuity of the vector field, which is sufficient for the problem of matching static CT-slices.

Having moving objects, there are discontinuities of the displacement vector field at occluding edges. Nagel and Enkelmann (1984, 1985) introduce oriented smoothness constraints to deal with this problem. Their approach suppresses continuity equally towards "inside" and "outside". One would wish to have a scheme, where information about the displacement flows from the edges of objects only to the interior.

This is actually possible, even on the basis of only primary image information. The idea of this approach is to limit the Laplacian of the membrane model to regions of the same label. One simple segmentation criterion is the sign of the Laplacian which is also used for the determination of the local correlations. One reason to choose this segmentation is that it is consistent with the choice of the operator to determine the local forces: the segment borders conicide with the places of the strongest forces.

Therefore in this model the zero-crossing lines of the Laplacian are the potential discontinuities in the displacement vector field.
The region-oriented Laplacian for the membran equation is constructed as follows:

$$\Delta_R U(X) = \sum_{X_i \in R} \{U(X+X_i)-U(X)\}$$

R is a local region with the same label as the central pixel.

The Dynamic Pyramid

The application of the simple membran model leads to two fundamental difficulties, that make an extension necessary.

First the convergence of iterative algorithms for sparse systems of equations is very slow, especially in big images containing large areas without external forces. Furthermore the range to find corresponding points by the described procedure is limited to the convex part of the local correlation signal.

Both problems lead to the use of pyramidal concepts. It has been shown (Terzopoulos 1983), that a surface interpolation scheme involving similar equations could be accelerated enormeously by applying a multilevel technique.

Additionally the distances are reduced by a factor 2 with each coarser level of the pyramid. By choice of a sufficient coarse level it is always possible to have the displacements between two consecutive images of a sequence within reach of local operators.

Within the concept of the Dynamic Pyramid the matching process is started at a sufficiently coarse level, where convergence is reached fast.

The results of this matching are transformed to the next level of the pyramid by a bicubical interpolation scheme and a forward image transform technique. These starting values for the next level are within 1 pixel from the final values at this level. Therefore also here convergence is reached fast and the search area is easily covered by local operators of the size 5x5.

This process is continued up to the fines level.

A sketch of the Dynamic Pyramid is shown in fig. 1. The correct match of every pixel at each level is essential for this process. It remains an open question, whether it is necessary to include error-correcting schemes in terms of "backtracking". If it turns out to be necessary, this can be achieved by transmission of information from "fine" to "coarse". The practical realization is limited to the flow of information from "coarse" to "fine".

Results and Discussion

The algorithm has been applied successfully to the "Taxi-sequence" from Hamburg, two frames of which are shown in fig. 2. The frames are first transformed into the binary Laplacian pyramid (fig. 3), which is the basis for the local correlation as well as the segmentation for the regionoriented smoothness constraint (Burt 1984). Three levels of the pyramid are used for this sequence.

At the coarsest level the displacements are within 1 pixel, and convergence of the matching is reached after 20 iterations.

Fig. 4 shows the displacement vector field at the middle level.

Fig. 5 shows the final matching results of the Dynamic Pyramid. The displacement vectors are more or less correct. The regionoriented smoothness constraint has the effect of a sharp cutoff at zerocrossing boundaries, but also results in a slight directional change within areas of equal motion. This effect is reduced by the fact that the zerocrossings of different levels of the Laplacian pyramid do not coincide except at real edges.

There are various directions of further development of the Dynamic Pyramid. One area of research concerns the local external forces, for which other operators can be included to reduce the problem of ambiguity. Also the segmentation for the regionoriented smoothness constraints can be done in many different ways. This work has shown, however, that simple operators combined with a powerful framework yield good results.

Acknowledgements

The author is grateful to Prof. Radig and Dr. Dreschler-Fischer from the Hamburg Institute of Informatics for providing the tape with the image sequence, and for the help and encouragement of the image processing group of the German Cancer Research Center, especially to Markus Schmidt for valuable discussions and for providing the forward image transform algorithm.

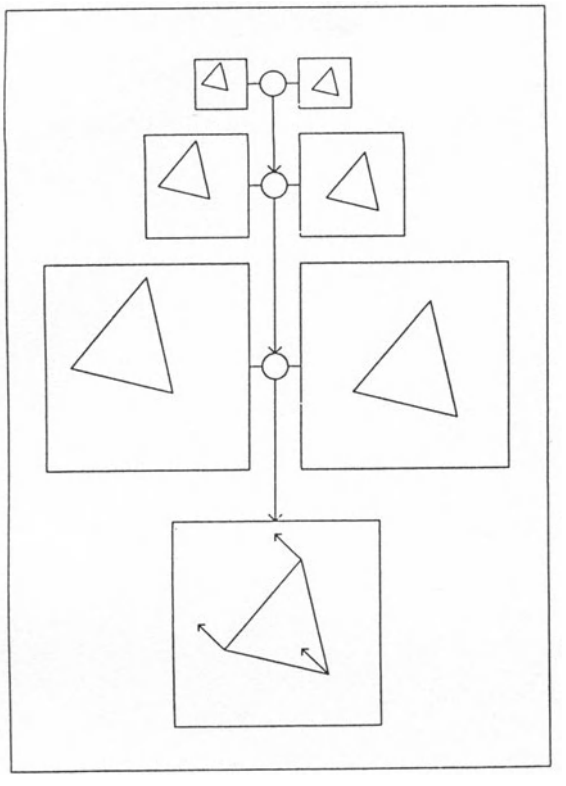


Fig. 1 The Dynamic Pyramid

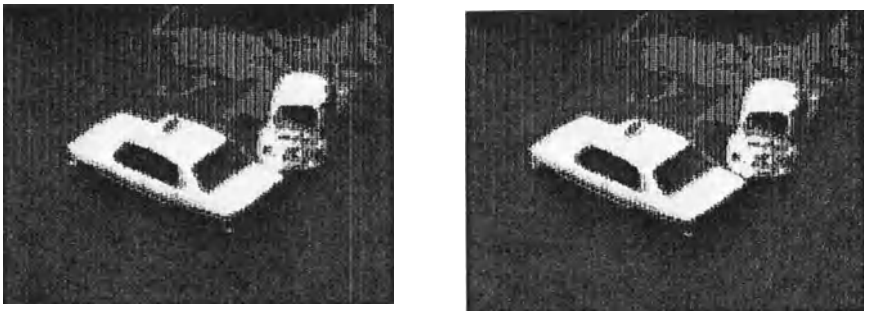


Fig. 2 Two successive frames of the sequence



Fig. 3 Binary Laplacian Pyramid

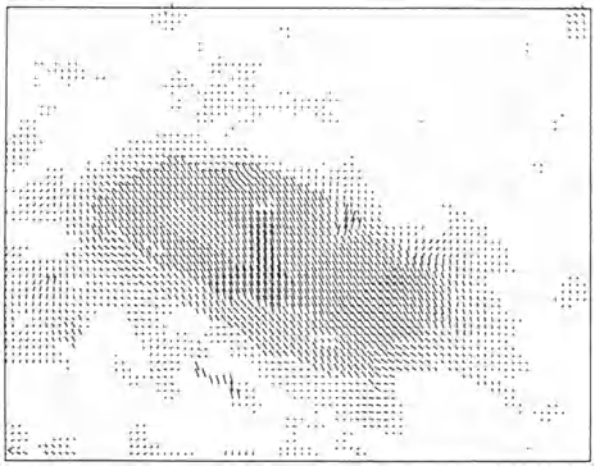


Fig. 4 Displacement vector field at level 2

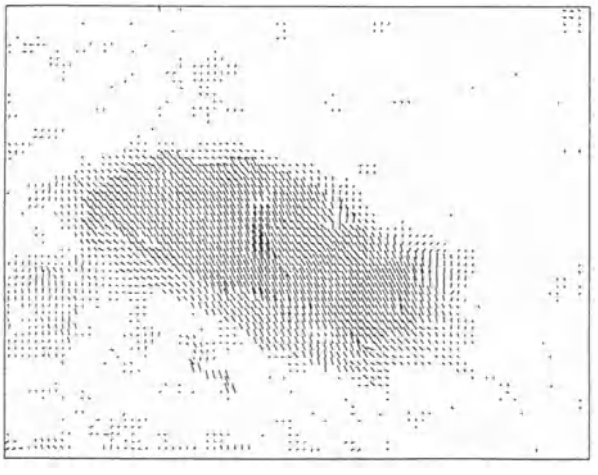


Fig. 5 Final displacement vector field

Literature

Bajcsy,R.; Lieberson,R.; Reivich,M. (1983)
A Computerized System for the Elastic Matching
of Deformed Radiographic Images to Idealized Atlas Images
Journ. of Comp. Ass. Tomography 7(4): 618-625

Broit,Ch. (1981)
Optimal Registrations of Deformed Images
Dissertation, Univ. of Pennsylvania, USA

Burt,P.J. (1984)
The Pyramid as a Structure for Efficient Computation
in Rosenfeld,A.(ed.),Multiresolution Image Processing
and Analysis,Springer,Berlin

Dengler,J. (1985)
Methoden und Algorithmen zur Analyse bewegter
Realweltszenen im Hinblick auf ein Blindenhilfesystem
Dissertation Univ. Heidelberg

Feynman,R.(1964)
The Feynman Lectures on Physics
Vol. II, Kap. 39.29
Addison-Wesley, Reading, Mass., USA

Hildreth,E.C. (1984)
The Computation of the Velocity Field
Proceedings R. Soc. Lond. B 221, pp. 189-220

Horn,B.K.P.; Schunck,B.G. (1981)
Determining Optical Flow
Artificial Intelligence Vol. 17, pp. 185-204

Nagel,H.H.; Enkelmann,W. (1984)
Towards the Estimation of Displacement Vector Fields
by "Oriented Smoothness" Constraints.
Proceedings 7th ICPR, pp. 6-8

Nishihara,H.K. (1984)
PRISM: A Practical Real-Time Imaging Stereo Matcher
MIT A.I. Memo No. 780, Cambridge, Mass., USA

Prager,J.M.; Arbib,M.A. (1983)
Computing the Optic Flow: The MATCH-Algorithm and Prediction
Comp. Vision, Graph.,and Im.Proc. 24, pp. 271-304

Terzopoulos,D. (1983)
Multilevel Computational Processes for Visual Surface
Reconstruction
Comp. Vision, Graphics, and Im.Proc. 24, pp. 52-96

Vectorial Features in Pyramidal Image Processing

H.-G. Zimmer
Institute for Microelectronics Stuttgart
D-7000 Stuttgart 80, F.R.G.

The notion of pyramidal data structures is useful for segmentation, classification, and representation of images. Ordinarily scalar features are calculated for a selected level of resolution and combined with corresponding features at different levels hierarchically after classification. As a generalization vectorial features are defined. Their components are taken from different levels of Laplacian pyramids and classified by similarity in a vector space which is homogeneous with respect to resolution. These features allow general or quite specific representations of image details.

Introduction

The transformation of signals to symbols is an important task in image processing. In the past many such procedures have been investigated or implemented with more emphasis on computational efficiency than on general applicability. Recent developments and trends in VLSI design and system architecture provide new aspects of computational costs. In the future the regularity of a systematic approach may become as efficient as a specialized solution optimized for a general purpose computer.

Therefore, the aim of the approach presented in the following is to define a complete set of numerical features which are invariant under translation, rotation, and scaling. Completeness and invariance are claimed with respect to the signal providing the image. It is required that the features allow the signal to be completely reconstructed and that transforms t of image coordinates corresponding to translation, rotation or scaling provide the same features $\overline{f}$ at the transformed coordinates: $\overline{f}(x,y) = \overline{f}(x',y')$ for $(x',y') = t(x,y)$. The features may be incomplete descriptions of real scenes, e.g. for the representation of depth, and they may vary with different aspects of a solid body due to 3D motion or changes of illumination. Furthermore it is desirable to have simple functions for calculating the features, in fact linear functions of the image data. This allows the treatment of the front end of image processing from data acquisition to feature extraction as a linear system and to defer nonlinearities to classification which is inherently nonlinear.

NATO ASI Series, Vol. F 25
Pyramidal Systems for Computer Vision
Edited by V. Cantoni and S. Levialdi
© Springer-Verlag Berlin Heidelberg 1986

Vectorial Features in Laplacian Pyramids

A digital image I(x,y) is given by X*Y nonnegative quantized intensities I at the integer coordinates $0 \leq x < X$, $0 \leq y < Y$. For simplicity we assume $X = Y = 2^m$, the general case can be treated by embedding the given image into a quadratic one which has a power of two as number of pixels per line. The intensities are band limited samples of a measured signal.

The Laplacian pyramid LP is constructed from the image data essentially in the way described by BURT (1) or CROWLEY (2). The frequency decomposition becomes very lucid if it is presented in the Fourier domain. The two-dimensional complex spectrum of the periodically repeated image is defined on a square in the frequency domain (Fig. 1). Its boundaries are given by the origin (frequency zero) and the Niquist frequency N. In order to get level LP_i ($0 \leq i \leq m$) of the Laplacian pyramid the spectral components of area i are transformed back into the spatial domain by an inverse Fourier transform assuming zero as spectral components outside the area i. (One half of the spectral components on boundaries between adjacent areas has to be used in both of these areas.) Level zero is a constant, the mean value of the intensities, and corresponds to frequency zero.

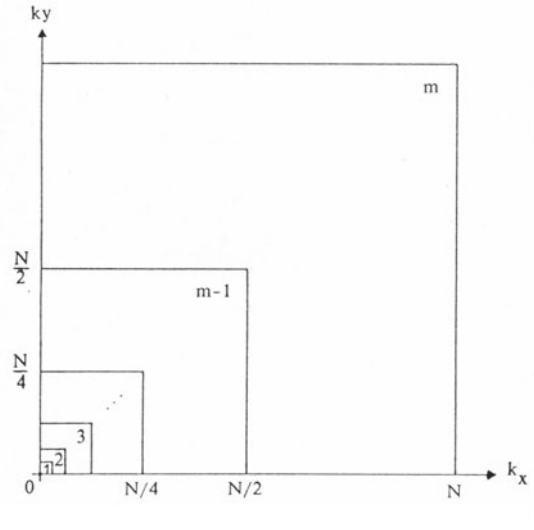


Fig. 1: Representation of frequency decomposition in the Fourier domain of an image. N denotes the Niquist frequency.

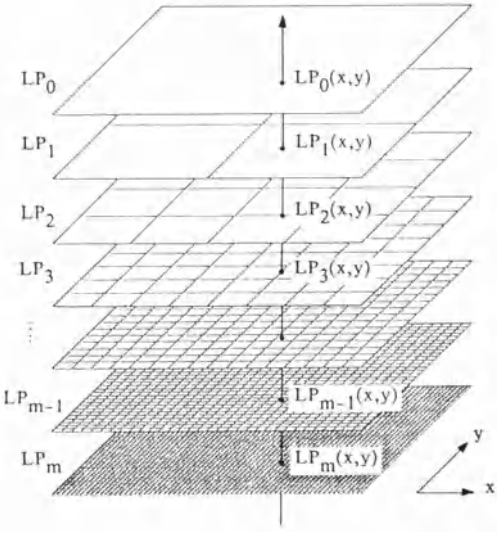


Fig. 2: Representation of frequency band decomposition in the spatial domain by Laplace planes LP_i. The values at the coordinate (x,y) provide the components of the vectorial feature $\vec{f}(x,y)$.

For 2^m pixels per line the descibed decomposition provides m+1 levels of the pyramid by always taking halves of the frequency range. But it is obvious that other schemes of decomposition or other types of band pass filters (e.g. partially overlapping) could be used as well. In fact the Fourier decomposition is computationally expensive and sensitive to violations of the prerequisites of the sampling theorem, e.g. aliasing or truncation errors.

The corresponding decomposition in the spatial domain is indicated by Fig. 2. The planes have the same size as the original image I(x,y), the grids give an impression of the "flatness" of that layer. In fact one sample is sufficient to represent the top layer, which is constant, four samples can represent the next layer, and so on. The full number of X*Y samples are required to represent the bottom layer LP_m. By reducing the number of samples in each layer to that minimum one gets the usual representation of a pyramid. But in the following it is conceptionally easier to treat all layers as equally X*Y sized sheets. Then the pyramids degenerates to a pile. The layers are no longer images in a strict sense because they may contain negative values.

The pile structure of image data suggests a natural way of mapping pixels into (m+1)-dimensional vectors:

$$I(x,y) \rightarrow (LP_0(x,y),LP_1(x,y),...,LP_m(x,y))^T = \overline{f}(x,y) \qquad (i)$$

T denotes the transpose of the row. This map is an embedding of the scalar intensity into a (m+1)-dimensional vector space of frequency band pass filtered data. For an image consisting of 512*512 pixels and the decomposition described above the vector space is ten-dimensional. The vector $\overline{f}(x,y)$ is called the *vectorial feature* of the image I at the position (x,y). It depends on the procedure of image decomposition and on the image data. But as long as linear filters are used for decomposition the vectorial features are linear functions of the image data: The sum of two images yields the sum of their vectorial features, multiplying the intensities by a common factor gives the same factor for all vectorial features.

According to the decomposition procedure all components $LP_i(x,y)$ of a vectorial feature $\overline{f}(x,y)$ add up to the intensity I(x,y) at that point. Since the intensities are a complete representation of the image signal, the vectorial features are complete in the same sense. In fact the image is isomorphic with the principal diagonal of the vectorial feature space, which is spanned by the vector $(1,1,...,1)^T$. Therefore, the representation of an image by its vectorial features is redundant. But the redundancy is reasonable. For any given vector with m real components there is an image having that vector as components 1 to m of a vectorial feature at an arbitrary point. Due to the linearity and invariance under translation it is sufficient to construct planes LP_1 to LP_m from suitable frequencies and the amplitudes at the origin given by the vector. Finally the component LP_0 is determined to ensure nonnegative intensities.

The advantage of the redundancy is that characteristics of differently sized neighbourhoods are explicitly included in a vectorial feature.

Due to the spatial homogeneity of the filters used for the decomposition the vectorial feature is invariant under translation of the object in the following sense: The translation is given by x'= x + a, y'= y + b for the periodically repeated image (or x'= (x + a)modulo X, y'= (y + b)modulo Y for a representation as a torus). Then $\overline{f}'(x',y') = \overline{f}(x,y)$ because the same intensities enter both calculations. An item moving relative to stationary items will not keep its vectorial features constant in a strict sense. The components from higher levels of the pile may change according to modified relations with other items. Lower levels (corresponding to higher spatial frequencies) are less sensitive and can be used as translational invariants in an intuitive sense of motion.

Invariance of vectorial features under rotation of the coordinate system requires circular symmetry of the filters used for decomposition. This is achievable. But it will be demonstrated by examples that even filters with quadratic symmetry provide an independence of angular position which may be adequate to many applications.

The invariance under scaling needs further explanation. The size X*Y of the image is considered to be constant. Scaling by a factor c means sampling the image with a step size of c units, thus covering only a portion ($c < 1$) or an area larger than the original image ($c > 1$). The scaled image has to be band limited too, which is difficult to achieve because of the truncation error at the boundaries ($c < 1$) or aliasing and truncation error for $c > 1$. In general the vectorial features will vary under scaling. But if $c = 1/2$ or $c = 2$ (or a power of two) all the components with the exception of the first and the last move up ($c = 1/2$) or down ($c = 2$) one step. Therefore, the vectorial features vary under scaling, but the effect is small for factors c close to one. Scaling by powers of two is explicitly described by a shift of components of the vectorial features. In practice the problem of size can be solved by calculating similarities including several components.

In summary vectorial features are:

linear functions of image data,
a complete but redundant representation of the image,
invariant under translation,
de facto invariant under rotation,
insensitive to small changes of size.

A vectorial feature is given explicitly by m+1 real numbers. Since pyramids are often used as a hierarchical data structure it should be noted that vectorial features are not hierarchical at all.

Similarity with regard to vectorial features

In order to measure similarities of vectorial features a metric is introduced into the (m+1)-dimensional vector space $\bar{f}(x,y)$. Using a weighted Gaussian distance the *similarity* $S(\bar{f}_1,\bar{f}_2)$ of two vectorial features $\bar{f}_1 = (f_{10},f_{11},...,f_{1m})^T$ and $\bar{f}_2 = (f_{20},f_{21},...,f_{2m})^T$ is defined by

$$S(\bar{f}_1,\bar{f}_2) = \mathrm{SQRT}(\sum_{i=0}^{m} w_i(f_{1i} - f_{2i})^2) \qquad \text{(ii)},$$

where the weights w_i are nonnegative real numbers. The weights are used to reduce or amplify the contribution of selected spatial frequency ranges. Since the distance is nonnegative the similarities between all vectorial features of a given image and a fixed or variable vectorial feature (e.g. taken from a second image) form a new image of the same size. Thus similarity maps the vectorial features of an image to another image. In principle the process of decomposition and calculating similarities may be iterated.

There are special instances of similarity useful in image processing. If the reference vector $\bar{f}_2$ is zero and all weights are equal to one, then the similarity is the norm of the vector space and gives at each pixel the "integrated local power", the sum of the squared amplitudes of all the frequency ranges used for the decomposition. If all but one weights are zero the similarity provides the power of a selected frequency range as it is used by LAWS (3) or ANDERSON *et al.* (4). By similarities the concept of local power is generalized for arbitrary subspaces of vectorial features.

An important property of similarities is their ability to describe structures or items in an image by comparing vectorial features with a given vector and within selected frequency ranges. A segmentation based on similarity is independent of position, orientation, or size in that sense described above for vectorial features. In contrast to usual procedures of classification within one level of the pyramid (e.g. CROWLEY (2) or HARTMANN (5)) this approach is in principle homogeneous over different levels of resolution. Of course, the homogeneity is impeded by the quantization steps of the frequency decomposition. The following examples will demonstrate that only simple structures can be identified in one step by their similarities. In general recognition needs a combination of several similarities. The limits of this approach still have to be investigated.

The components of a reference vector and the weights selecting the frequency range can be determined from the knowledge of the spectral properties of the desired structure. But in practice it is more convenient to look at samples of that structure. A scan through the norm of subspaces with different frequency ranges shows the discriminating power of frequencies. Knowing the coordinates of a point of interest the components of the reference vector for a suitable frequency range can be read off from the pile or the pyramid. We only used weights

equal to one or zero.

Two images can be compared by calculating the similarities of corresponding vectorial features in both piles or pyramids. This allows the detection of motion regardless of low frequency changes (e.g. due to illumination) and high frequency disturbances (e.g. noise or camera vibration), or to determine the fixed point of rotation.

Examples

In the following examples the digital images were obtained by scanning an area of 51.2mm *51.2mm of transparencies with a step size of 0.1mm. Band limitation was achieved by a circular measuring diaphragm with a diameter of 0.24mm. The intensity was quantized to twelve bit words. The scanner and its characteristics are described by KRONBERG *et al.* (6). In all cases with the exception of Fig. 8 the band pass filtering for decomposition was done in the frequency domain using a fast Fourier transform FFT. The presented pictures are reduced photographs of hardcopies produced by a SORO laser plotter.

Fig. 3 shows the image PATTERN which provided the data for Fig. 4 and Fig. 5. Fig. 4 shows the norm of selected subspaces calculated as similarities $S(\bar{f}(x,y),0)$ for two adjacent levels. Fig. 5 shows the similarities $S(\bar{f}(x,y),\bar{f}_i)$ in subspaces for selected vectorial features $\bar{f}_i$ as described in the captions. Basically the same is presented by Fig. 6 in order to demonstrate the sensitivity to changes of size and the response to complex structures like the

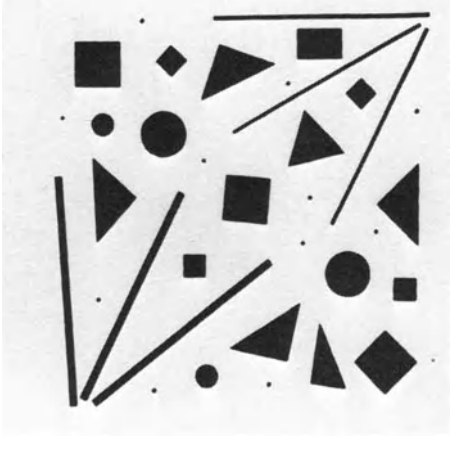


Fig. 3: Digital image PATTERN consisting of 512*512 pixels. Band limited scan of a transparency, intensity quantized to twelve bit words.

windows. Fig. 7 is an example of a noisy picture and demonstrates the invariance of similarity under orientation. The similarities in Fig. 8 are calculated using a pyramid and linear interpolation for intermediate terms instead of a pile. The artefacts at the rim of the lead straps indicate that linear interpolation is not satisfactory.

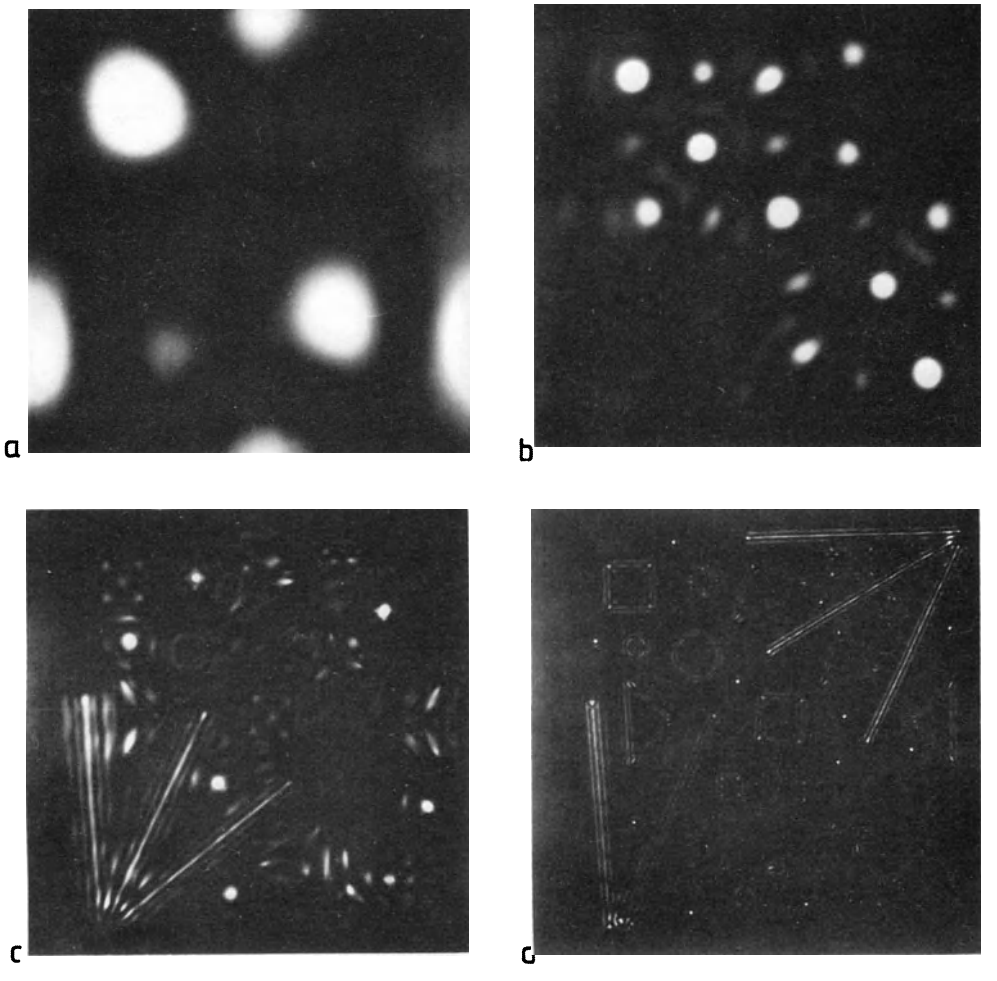


Fig. 4: Norm of selected subspaces of image PATTERN calculated as similarities $S(\bar{f}(x,y),0)$ for the following two levels: **a)** Levels 1 and 2, **b)** levels 3 and 4, **c)** levels 5 and 6, **d)** levels 7 and 8. Note: Level 0 is constant, the mean intensity of the image.

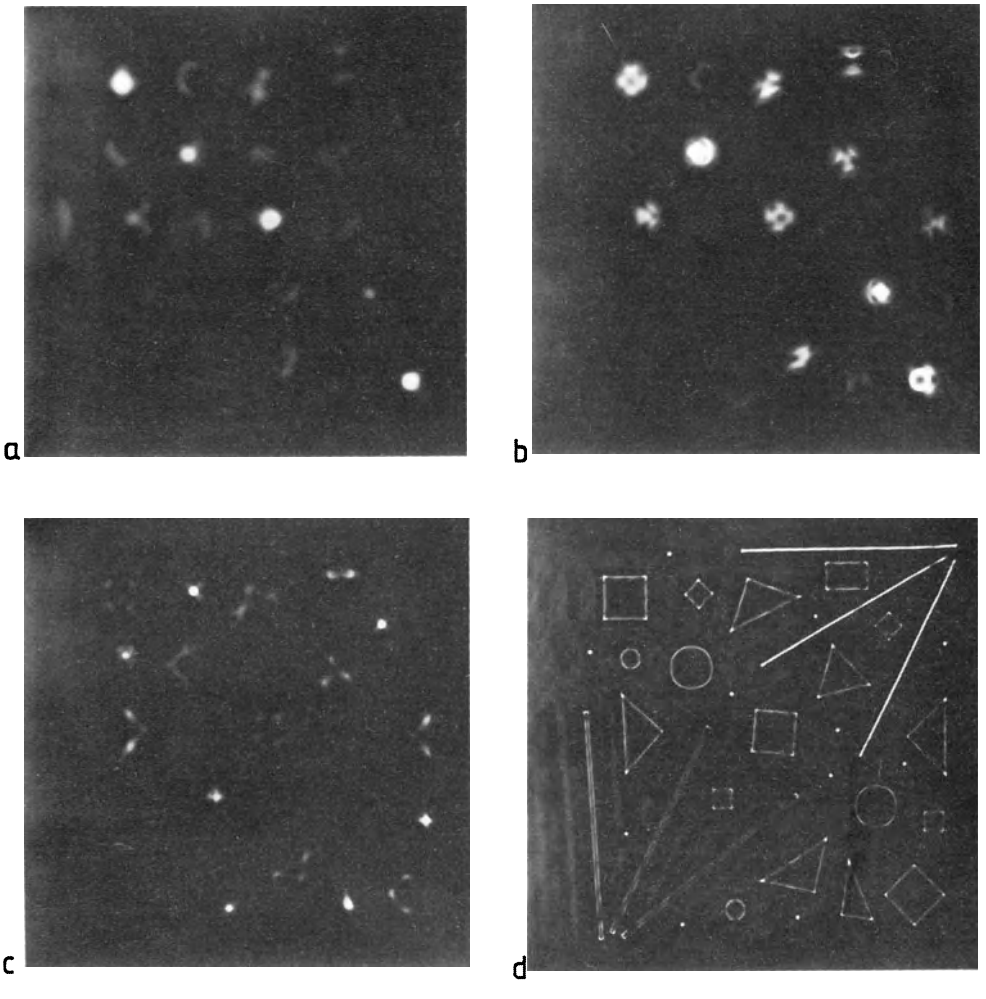
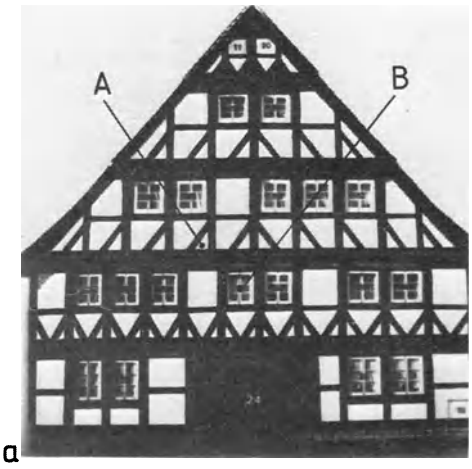
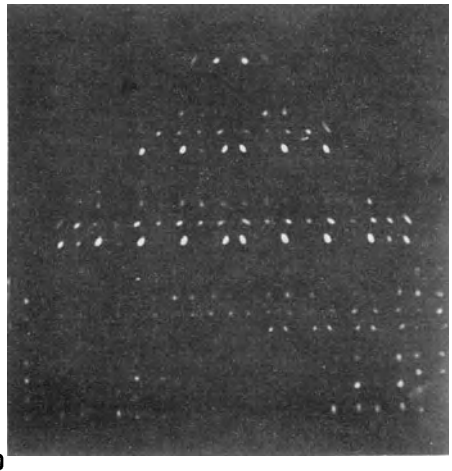


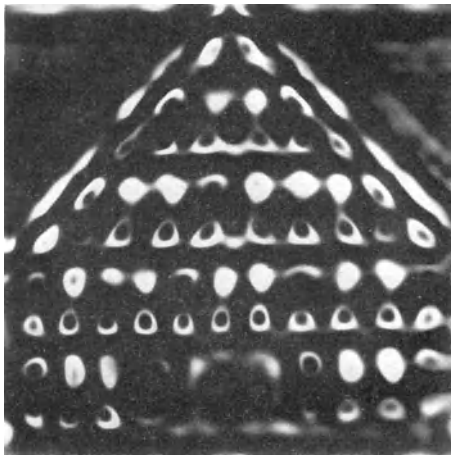
Fig. 5: Similarities $S(\bar{f}(x,y),\bar{f}_i)$ in subspaces of image PATTERN for selected vectorial features $\bar{f}_i$. **a)** Components of $\bar{f}_i$ taken from the vectorial feature at the center of the upper left large square and at levels 3 to 6. The weights for the other levels are zero. Similarity zero describes a perfect match and is displayed as white. Similarity is in fact independent of position or orientation of the squares. Among other structures the large circles have the best similarity, but can be discriminated by thresholding. **b)** Components of $\bar{f}_i$ taken from the center of the upper large circle and at the same levels 3 to 6. **c)** Same components as in a) but now used for levels 4 to 7. This means looking for similarity to squares with one half of the linear extend. Here similarity does not give a clear discrimination between the small squares and the triangle in the lower right corner. **d)** Components of $\bar{f}_i$ taken at a point on a narrow beam and at levels 5 to 8. Good discrimination against wider structures but no difference to similarity at small dots and insignificant differences at corners of polygons. A combination of c) and d) allows the detection of small squares and the discrimination from triangles.



a



b



c

Fig. 6: Similarities in subspaces of image HOUSE. **a)** Image with arrows pointing to positions A and B used as references for calculations of similarities. **b)** Components of reference vector $\overline{f}_i$ taken at position A (triangle of framework) at levels 5 and 6. Note that the triangles in the lower row are of different sizes which becomes obvious in the similarity picture. **c)** Components of $\overline{f}_i$ taken at the center of a window (position B) at levels 2 to 5. The similarity picture still "shows" the windows but does not discriminate them against other structures by thresholding.

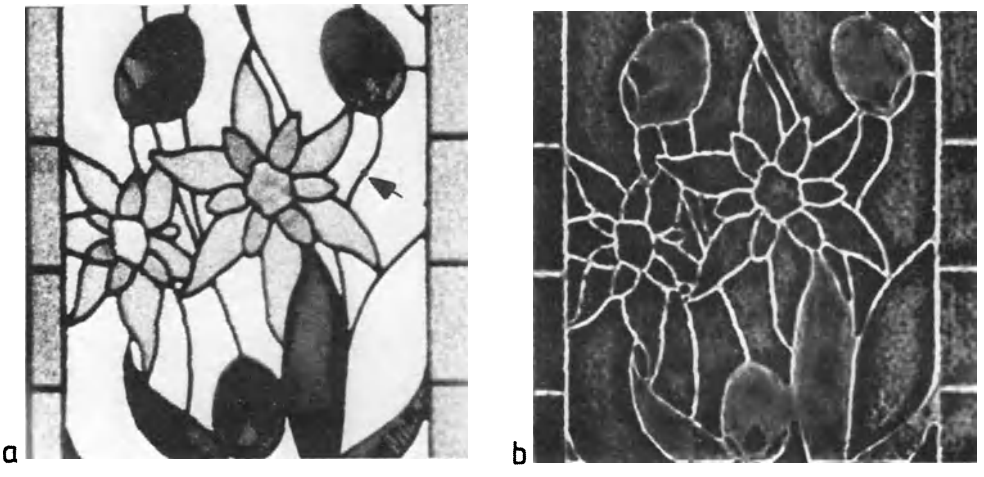


Fig. 7: **a)** Image FLOWER as an example of a noisy picture. The position of a reference vector is indicated by an arrow. **b)** The reference is taken at the middle of a lead strap at levels 5 to 7 in order to demonstrate the invariance under rotation.



Fig. 8: The image FLOWER was used to build a pyramid instead of a pile by reducing the levels to the necessary number of samples. For the calculation of similarities intermediate terms between the samples have been determined by linear interpolation. The levels are the same as for Fig. 7b, the position of the reference is nearly the same, but on the rim of the lead strap in order to demonstrate the artefacts clearly. In fact the pyramid was built by the differences of a low pass filtered level against that level. But the effect of interpolation does not depend on the procedure of decomposition.

Fig. 9 shows motion detection as an example of similarities between two images. By restricting the analysis to relevant levels slow changes of the illumination and short range mismatches due to vibrations of the camera do not influence the result. Extending the analysis over several adjacent layers is a generalization of the approach presented by ANDERSON *et al.* (4).

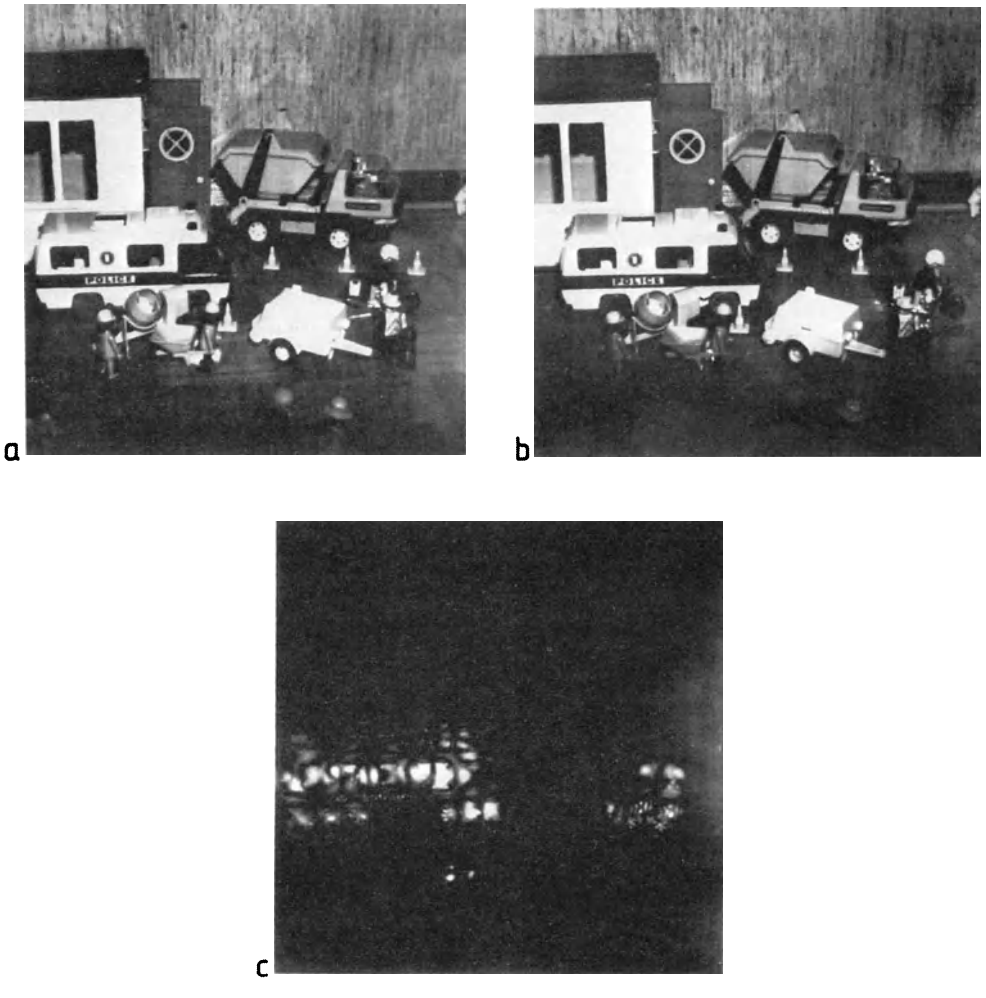


Fig. 9: Motion detection as an example of similarities between two images. **a)** and **b)** are two frames of a sequence showing the movement of a car and a motor-cycle, **c)** shows the similarity $S(\bar{f}_a(x,y),\bar{f}_b(x,y))$ calculated for the levels 5 to 7. In contrast to the preceeding pictures the similarity zero is displayed as black.

Conclusion

Vectorial features have been defined using a pyramid or a pile structure of image data. They provide a complete but redundant representation of an image, are linear functions of the image data and are invariant under translation, de facto invariant under rotation and insensitive to small changes of size. Similarity is defined by distance of features in a vector space. It provides information about selected features as an image and thus allows the localization or classification of corresponding image structures. The main reason for introducing vectorial features is the opportunity to treat an image as a linear system in an investigation combining coarse and fine structures. A consequence of that approach is a large amount of calculations and intermediate results. But the structure of the algorithms is very regular and suitable for multiprocessor systems.

Professor V. Neuhoff and the Max-Planck-Institute for experimental Medicine at Göttingen supported this work by providing the image data and the SORO plots. Their help is gratefully acknowledged.

References

(1) BURT, P.J.: The Pyramid as a Structure for Efficient Computation. In: ROSENFELD, A. (Editor): Multiresolution Image Processing and Analysis. Springer Berlin Heidelberg New York Tokyo, 1984, pp. 6 - 35

(2) CROWLEY, J.L.: A Multiresolution Representation for Shape. In: ROSENFELD, A. (Editor): Multiresolution Image Processing and Analysis, Springer Berlin Heidelberg New York Tokyo, 1984, pp. 169 - 189

(3) LAWS, K.I.: Texture Energy Measures. In: Proc. DARPA Image Understanding Workshop, Los Angeles, 1979, pp. 47 - 51

(4) ANDERSON, C.H., BURT, P.J., VAN DER WAL, G.S.: Change Detection and Tracking Using Pyramid Transform Techniques. In: Intelligent Robots and Computer Vision, Proc. SPIE Vol. **579**, 1985, pp. 72 - 78

(5) HARTMANN, G.: Erzeugung und Verarbeitung hierarchisch codierter Konturinformationen. VDE-Fachberichte **35**, VDE-Verlag, 1983, pp. 378 - 383

(6) KRONBERG, H., ZIMMER, H.-G., NEUHOFF, V.: A High-Performance 2D Gel Scanner. Clin. Chem. **30**, 1984, pp. 2059 - 2062

PROGRAMMING IMAGE PROCESSING MACHINES

S. Levialdi
Università di Roma
Dep. of Mathematics
00185 Rome, Italy.

Introduction

The development of image processing systems from sequential machines started by changing their periphery to acquire and display pictures; later multiprocessor systems where introduced which could operate in SIMD/MIMD mode and, consequently, an evolution in the programming languages to be used was mandatory. On the other hand many other application areas have found the need to improve both the human-machine communication and the cost-effectiveness of the programming effort. New ideas coming from human perception and the proposals of new programming paradigms have also modified the high level languages commonly employed on computers so as to blurr the separation between a compiler, an editor, an interface, an environment. These new approaches will also have an impact in the future languages that will be used for programming non-Von computers for vision such as the pyramid systems discussed in this Workshop.

Historical notes

The human effort devoted to the design and testing of new languages for image processing (on sequential single processor or on multiprocessor machines) may be estimated of the order of 100 man years, on a time span of only 35 years, by extrapolation from[1] and concerns the work performed to design about forty image processing languages. The influence of the Fortran language is very strong except for those machines having a particular set of processors, of varying complexity, which must be controlled and where therefore the need of parallel constructs is mandatory. A number of typical operations (listed in[1]) shows that no existing language is able to include, at a primitive level, all the necessary image manipulation operations which range from input-output to particular computations for performing classification, spectral analysis, etc.

The need for efficient, interactive yet powerful image processing systems was

NATO ASI Series, Vol. F 25
Pyramidal Systems for Computer Vision
Edited by V. Cantoni and S. Levialdi
© Springer-Verlag Berlin Heidelberg 1986

clear from the beginning but the availability only of sequential machines played also a part in the generation of particular software packages which were essentially a number of subroutines, callable by a Fortran main program, which could perform the requested operations on the images to be processed. There were also special packages which simulated the availability of a parallel machine to the sequential machine user[2] so that many constructs could equally well apply regardless of the architecure of the machine at the expense of execution time. The architecture of new machines and the evolution of high level languages (from Fortran to Algol and later to Pascal within the area of scientific computation) played a major role in the language to be chosen or extended. Some examples of suggestions for language implementation and real languages that have been in use on image processing systems may be found in[3]. Typically, extensions of Algol[4] and Pascal[5] have been proposed and, more recently, extensions of C and even the use of Lisp for some specific image processing tasks.

The problems faced by the computer architects were complex enough to distract them from thinking about the language issues to be faced to allow comfortable programming of their machines; in fact most projects started from the consideration of the high throughput requirements needed for practical image processing and turning then to a real multiprocessor system project and then to the design of the single processor. The day-to- day problems to obtain the VLSI chip manufactured and the choice or invention of clever algorithms matched to the specific machine were a burden that largely overcasted the programming tools specification, design and construction.

In fact even in the MPP[6] (Massively Parallel Processor) project the language came next, after the hardware design, and only recently a Parallel Pascal which may run on it has been designed, tested and is beeing used. Efforts concentrated on portability of algorithms, on using a special "matrix algebra" for SIMD architectures and on utilizing all the available processors of the specific architecture. As usual, generality opposed efficiency and only those algorithms taking the specific SIMD architecture into account were able to exhibit some efficiency (as an example see[7] where the GLYPNIR language is designed on the basis of the pattern articulation units of the ILLIAC IV machine).

A recent classification of computer languages[8] shows that if their level (or man-machine distance) is considered, the base of a triangle represents the dominion of the machine languages (very low level) and, gradually climbing upwards, the low and intermediate level languages, the high level procedural languages, the menu-driven programs, finally only one executable program which may be triggered by a single button. In this classification the emphasis is on the possibility of having new programming tools, namely those introduced by icons (like those in a Macintosh environment) that may relieve the programmer's effort. This approach, of icon-oriented programming, has never been applied to image processing or analysis nor to multiprocessor machines. Yet there are

some principles, implied in visual programming, which may turn very useful in designing new languages and environments for the friendly, error-free programming of image processing tasks.

Many other programming paradigms may be considered (apart from procedural and visual) such as functional, logical, object-oriented: in fact a great deal of attention is given today to the subject of programming languages, to their environment and on the nature of human interaction[9,10]. The key issue is whether one should attempt to use many different compilers in one unified environment (like C, Pascal, Lisp, etc. under Unix) or use/develop a "Universal Programming Language"[11](UPL) to match both the application areas (classes of algorithms) and the chosen computing architecture. Problems of side-effects and difficulty in mastering all the included languages counterbalance the effort of designing a new language from zero or of extending an existing language (or adapting a well known one) to the needs of the image processing user.

Another approach, which has been addressed by some researchers, is that of integrating the image processing system (whether uniprocesso or multiprocessor, for research or production, sequential or parallel) into a global pictorial information system which may, according to the requirements, acquire, store, retrieve, manipulate, display, print, images of various formats such as those obtained by different sensors.

In conclusion, image processing languages, like all computer languages, are sensitive to two opposite forces: one trying to improve the existing software tools (languages, environments, operating systems) to allow convenient programming of the image information system; the other one, attempting to define a new, portable, efficient, transparent and modular language which may run on currently existing machines. However, the influence of Artificial Inteligence will certainly encourage to use Lisp (or Prolog) so that, for instance, some objects (formally defined by rules) present in the stored images may be automatically retrieved via an inferential process.

Language extensions

In a standard description of a well known subroutine package[12] for image processing the different tasks that may be executed by the contained programs are subdivided into the following areas (the number in brackets corresponds to the different programs which perform the same task with slightly different algorithms):

Orthogonal transforms (53), these are very important and contain typically the Fourier Transform, different convolutions, Discrete Cosine, etc. Registration operations (26), generally used in computer cartography, Enhancement and Smoothing (16) for image cosmetics, restoration (15) for compensating

distortions introduced by the image acquisition instrument, edge and line detection (25) for contour and man-made objects extraction, application of relaxation labeling (13) used for image processing under a general relaxation model. Texture analysis (23) for the extraction of quasi-periodic structures identifying objects or materials, region segmentation (6) for partitioning the image into meaningful components, geometry (81) for shape description and analysis of digital shapes, basic operations (83) for performing statistics on the image, gray scale manipulation, expansion, contraction, etc and a final class of miscellaneous (9) where pattern synthesis and printing may be obtained.

As may be seen from this list of operations, they change in complexity, in number of parameters considered, in required accuracy (in some cases integers are sufficient in others floating point numbers are required), etc. Generally these operations have been classified in the literature as point operations (when they consider only one single pixel), local operations (typical of image processing and the main reason for encouraging array processing systems), global operations (like a Fourier Transform) and statistical (like histogram computation). On top of all these operations we are generally faced with the problem of the pictorial data base management (storage and retrieval) and, perhaps, the manipulation of pictorial information by semantic value (and not by memory address like in the case of number matrices), it is for this reason that the "language problem" for choosing a suitable formal language for image processing (and retrieval) has a number of radically different approaches and tentative solutions.

The main image processing centers which must perform bulk image enhancement and recognition (for well specified tasks like metereology, astronomy, land management, etc) use large sequential computers and Fortran-like languages which may call one and all of the routines (mentioned above) from commercial packages.

Some years ago, along the line of high level language extensions we have started a project[13] for the implementation of an image processing language (PIXAL) which could both be used on a sequential machine and on a cellular processor array so that parallel algorithms could be tested and their sequential (and parallel simulated) execution time could be experimentally found. The starting language was Algol which used structured blocks and a wide range of data types but the main innovations came from the required control structures which could differentiate between local parallel processes and global ones.

Typically, a local operator requires the updating of each pixel value (on the basis of some operation involving the neighbours) which, in parallel, means the simultaneous (and independent) computation of the new value of every pixel. The tool that was used for allowing the execution of these local operators was a new construct (**par-parend**) which simulated the parallel execution (on every element of each array appearing within these statements) of all the contained

instructions without requiring the typical double nested loops through the row and column indexes. On the other hand, it is sometimes useful to test the contents of an array for deciding whether a process should continue or stop or, in some other cases, it may be useful to assign a value to all the elements for the array; for these purposes the arrays (images) are simply called by their names and all their elements are implied at each operation so that for testing or assigning a value the = or := signs are used at the right hand side of the array's name.

New data types like **binary**, **grey**, **boolean** and special elements like **top**, **first**, **last**, **bottom,** and **edge** were defined to use the consistency tests on variable values and some specially useful data structures (like **frame** and **mask**) which allow the selection of a subset of pixels from the image and the choice of coefficients for convolution and filtering. Moreover functions (like **overlap** for logical masking and **overweigh** for performing local products) have been defined in order to be conveniently used in large collections of image processing algorithms.

Following the PIXAL syntax clear and transparent programs may be written and therefore one of the main purposes of using a high level language for image processing has been satisfied, yet the cost of this transparencey and generality (machine independence) is the lack of efficiency and of special instructions which take advantage of the existence and format of registers (like **gather** and **scatter**[14], for instancce) of activating interprocessor communication paths, of using all the available parallel resources. The PIXAL programs look nice and are easy to write but do not solve the general programming needs of parallel machines; in fact both the vector computers and the pipeline computers have special instructions, which are particularly tailored to their architecture, in order to allow the convenient execution of parallel instructions.

Along this line many other projects have expanded high level languages (like Fortran or Pascal) or used existing languages (with particular facilities for parallel processing) like APL. In some specific scientific centers it is very difficult to change from one language to another since many programs exist and the cost of rewriting them is obviously too high (this is similar to the situation of business applications and programs written in COBOL). In astronomical image processing, for instance, the trend is still to use Fortran but to move over to the Unix operating system and make use of new standards in communication protocols, bus lines, etc. It is thought that the standards, far from hindering new applications or ideas, foster the production of programs which may be easily distributed, tried and used.

Integrated systems

A completely different approach to the design of high level image programming languages considers the information processing system as a whole and therefore

tries to integrate, within the software environment, all the tools that may turn useful in the different operations to be performed on the images ranging from acquisition to display through the retrieval, the interaction, the manipulation and transformation. According to this view[15] the pictorial data base is also considered (stressing the importance of retrieval of images -containing certain objects-) as well as the possibility to deal with different representations of images (raster images, vector graphics, icons, codes). The main reason behind this approach has been to investigate what is lacking in existing languages more than the concrete proposal of a universal high level language for generalized image processing. This language (called IPL[15]) allows the user to navigate through the image database and manipulate images using generalized icons (which are intermediate representations of images having both a visual representation and a formal description). The IPL language is made of three components: the logical one (LIPL), the interactive one (IIPL) and the physical one (PIPL) acting on the real pixel-based images.

The central idea of IPL is to allow the user of the system to navigate comfortably in the data base by means of user-defined icons, windows and ports. The image data base, as in many other projects, is hierarchical in nature and may be enriched by user-defined images by the explicit association of parts of existing images. Generally the user will request detailed visualization of portions of images (specified by frames) so that the implementation of this facility is based on the concept of workspace (which must be mantained and updated to store the current entities of interest) which is equivalent to that of desktop in office automation systems.

The ports (communication channels) are used to enable the workspace to receive information from the database (and viceversa) or from peripheral units. The currency concept in IPL is taken from the currency concept in CODASYL[16] where the last accessed entity (of all definable types) is maintained keeping the types available in the workspace. Currency entities are all available allowing references to images, regions, objects, etc but instead of considering the present record as the last in a sequence, IPL uses a relational approach where operators like **next** or **prior** are unnecessary. A currency entity can be set, moved, opened and deleted, the currency command is indicated with **!**.

The main feature of IPL comes from the introduction of icons (a recent concept which is receiving an increasing attention by computer scientists designing new programming languages exploiting visual feedback and perceptually significant computing metaphors). The icon is a symbol representing an object (or a set of objects) belonging to one of the three following classes: a) an object icon (for example a port icon), b) a window icon (for an image viewing window) and c) a command icon referring to an operator within a command language. There are three ways in which an object icon may be recalled: by naming it, by pointing at it (with a mouse or with a cursor), specifying a rule defining the object set.

An object icon may be specified by naming it (or the object type) and by specifying a rule defining its corresponding object set, generally the object icon is associated with an icon port for its materialization (reproduction in data base terminology) so allowing to load the icon into the workspace via the chosen icon port. Window icons enable visualization and zooming into any other part of the image. Windows (and region frames) may have one of five possible types according to their shape: rectangle, sector, circle, corridor and polygon. These window shapes have been chosen for extracting specific information from maps, the default window type is a square (subtype of type rectangle). Windows also have a number of attributes which specify their location, extension and orientation.

Seven different basic commands may be applied on windows (and region frames): create, delete, enter, move, rotate, expand and contract. A second command (if meaningful) may be applied so producing a number of different presentations of the image as shown in the following table (taken from[15]).

Window

Region-Frame	Hold	Expand	Contract	Move	Rotate
hold		-	enlarge	reduce	shift
w-rotate					
expand	zoom-out	expose	-	-	-
contract	zoom-in	-	conceal	-	-
move		scroll	-	-	scan
-					
rotate	r-rotate	-	-	-	corotate

There are also other attributes that a window may have like 1) priority, 2) opaqueness, 3) visibility and 4) frame-visibility. These attributes are necessary since many windows may be present on the workspace at the same time; (1) establishes which window will stay on top, (2) will determine inter-window overlap, (3) will change to invisible those windows that have (1) under a certain threshold and, finally, (4) will enable/disable the frame visualization.

The icon ports are also driven by commands (create, delete, open and close) or activated by cursor commands like port.cursor. In order to achieve system protection and reliability a capability addressing mode is associated to the port by means of three status values: closed, open-to-update and open-to-read.

The main principles underlying the image retrieval process imply the notion of logical images which are a collection of picture objects and relational objects.

The picture object is a set of attribute triples (a-name, a-value, eval-rule) while a picture object class is a set of picture objects having the same type since a picture object class always has a unique type.

For instance, a picture object of type "river", whose name is "Tiber" and whose physical picture attribute is pp will be denoted by

v = {(type, "river", eval-rule), (name, "Tiber", eval-rule), (pp, *, eval-rule)}

The physical picture type pp is a pointer; the physical picture may be obtained by invoking the eval-rule which retrieves the corresponding physical picture from an image store and shows it to the user; the other two eval-rules corresponding to the other attributes presents the attribute values as strings, if the eval-rule is unspecified it is defaulted.

In a limiting case picture objects may correspond to pixels and for the type pixel having coordinates xi,yi ; name x,y and value f(xi,yi) we will have the following representation

vi = {(type, "pixel", eval-rule), (x, xi, eval-rule), (y, yi, eval-rule),
(c, f(xi,yi), eval-rule)}

The evaluation rule may present the pixels either in numerical form (typically positive integers including the zero value) or in gray-level form.
The evaluation rule may act when an attribute value is specified so as to interpret it and present the object to the user or it may act interpreting virtual attributes (unspecified attributes) as for instance when these attributes must be computed after an image processing algorithm has completed its task.

For instance the name attribute may be defined as a virtual one composed of other attributes as in the case of the pixel object defined as the combination of the two coordinates (x and y)

v = {(type, "pixel", eval-rule), (name, - , equal(x,y)), (x, 10 , eval-rule),
(y,2,eval-rule), (c, 5, eval-rule)}

The name is therefore a virtual attribute. A relational object has three attributes (name, value, rule) and the name may also be a virtual attribute defined in terms of other attributes. Sometimes the name is specified by a list of couples (type, name) for picture objects satisfying that relationship. A relational object class is a set of relational objects having the same type like, for instance, all the relational objects having the same type "over" will be grouped into one relational object class.

To retrieve information from an image database a query may be specified as a

conjuction of logical clauses relating attributes and their values, to retrieve the brdige over the river Tiber

< _w.bridge $ _bridge OVER Tiber.river

where w.bridge stands for {w:w.type = "bridge"}
and Tiber.river stands for {v:v.name = "Tiber" v.type= "river"}
$ stands for "where"

The same query may be also formulated as an inference rule like

?_OVER (_X.bridge Tiber.river)

The rules are used as in Prolog where a unification process will proceed to substitute terms for variables to make expressions identical. The rules can be applied in sequence (the system may backtrack), they are deterministic, may perform exception handling, they do not differ from data (data and rules are treated in the same way).

The Prolog approach uses Horn clauses (first order predicate logic with elementary assertions) which may not be easy to use when expressing relationships between components of two dimensional (or three dimensional) pictorial data. For this reason a high level notation has been introduced, the rules will then be "compiled" into a machine-oriented level where the code can be executed directly. As an example we describe the query to find a road which crosses the starting point of the segment of the current road in the current window, if more than one road is found then get the widest road:

? _ CROSS_ROAD(_X.road);
MAXIMUM(width_X.road)

only one road object is given here as a parameter to CROSS-ROAD although there are in fact two (that will produce a cross-road) but one of them is implicit, the current road, _X is the object variable of type road and width is an attribute of roads. If we want to find all roads that cross road R1 in the current window we will ask

?_CROSS_POINT(R1.road_X.road)

the answer to this query will give a name (of the crossing road) and the coordinates of each road which crosses R1 in the current window.

In conclusion, to define a rule one must give a rule name, the parameters and their types and the attributes of the different object types used in the unification process; finally the rule will be expressed either in terms of some programming

language including the name of the objects which should be returned and in some cases also the coordinates of the objects.

The IPL language not only provides the pictorial data base storing and retrieval mechanisms but also contains the graphical and manipulative components which are generally known as image graphics and image processing. A picture is a data structure representing an image and six types of pictures may be taken into account: logical area, logical line, logical point, physical area, physical line and physical point: the first three are used for representation purposes (image graphics) while the last three are the real images to be processed for quantitative analysis and interpretation.

The materialization (and dematerialization) operators allow the representation conversion among these six picture types while the image transforms which are generally useful may be grouped as point, local, statistical and global operations according to the number of pixels involved in each single transform required to produce the new (transformed) value of the pixel.

A general purpose, high level, construct called COMP* has been introduced which will identify, according to the value of *,one of the four classes COMPP, COMPL, COMPS and COMPT respectively.

As an example the computation of the arithmetic mean of a given set of local neighbours can be expressed as follows:

COMPL(picture_1 MEAN(picture_2))

which indicates the computation of the mean of the neighbours of each pixel in picture_2 and the substitution of these computed values in all pixels of picture_1; MEAN may be a user-defined function or a predefined function stored in the system library. Another example is the identity test between two images

COMPP(object_1 EQUAL(picture_1 picture_2))

object_1 will be true (single value) if and only if picture_1 is identical to picture_2.

If only one picture argument is specified for the operator EQUAL the default option is to consider the picture in the workspace as the other argument.

In many image processing applications a number of iterations must be performed and the stopping condition is data dependent, in order to express this for a general filtering operation we may write a program demonstrating the control structure visibility

```
while temp = FALSE
    begin
       COMPL (picture_2 FILTER(picture_1));
       COMPL(temp EQUAL(picture_1 picture_2));
       COMPP(picture_1 picture_2));
    end.
```

The picture_1 is filtered to create picture_2, the value of temp is set to TRUE only if picture_1 and picture_2 are identical and FALSE otherwise; finally picture_2 is copied back into picture_1. This program performs noise removal by local filtering (FILTER) until no further changes are produced on picture_1.

In conclusion the COMP* construct invokes the execution of the named function on all the elements of the named object (the image is not seen as a collection of pixels but as a whole object); COMP* (result function(image)) will compute on all elements of the object-image.

This approach integrates the different functions generally required in an image system such as the management of a pictorial database, the display of image components and the manipulation of images by means of a wide set of transforms. The host languages are typically C and Prolog (or Pascal and Lisp) and therefore at least two different languages will be used for implementing IPL which may be seen as a testbed to try new ideas about the programming needs of image information systems.

Visual languages

A few years ago a new series of IEEE Workshops was initiated (1984, Hiroshima, Japan) with the hope of radically improving the man-machine interaction in particular with respect to the programming effort. The lower cost of workstations having high definition and window management software and the universally recognized human feature of being able to understand new, complex concepts visually rather than linguistically were the driving forces of a new kind of programming languages: the so called visual languages. An introduction to the field of iconic systems may be found in [17] whilst the first collection of papers on this subject is in[18].

There are many ways in which a visual representation of the programming activity may be incorporated and they are all potentially useful in the way new programs (for image systems of for any other information system) may be written (or better, perhaps, drawn). The program may be visualized (as a number of pre-defined icons), the data structures involved may be seen on the screen, the output may be monitored graphically or a combination of these three possibilities

(and their evolution during program execution, sometimes called program animation) can be conceived

More specifically, we can imagine a three button mouse that allows the choice of program components (constants, variables, structured data types, of constructs, of procedures) within an environment containing a syntactic editor (ensuring correctness). When the choice has been made and a fast interpretation of the new born program has occurred we may then switch to compilation (for speed and efficiency on the specific machine) so as to obtain a final tested product: the program.

A classification of visual languages[19] is the following one where a distinction is made between languages **for** images and languages **by** images:

Visual Lang.	*Objects*	*Transforms*	*Visibility*	*Applic. areas*
Visual prog. graphics,	non visible objects	(Xm,e) ==> (Xm, X'i)	visually rep. constructs	CAD, VLSI
Lang. for images	visible obj. with logical represent.	(e, Xi) ==> (X'm,Xi)	non-visible constructs	pictorial data-base, office automation

As seen on the table the first line describes the programming languages which make objects visible in order to help the user in establishing relationships and produce configurations to be displayed, the second line describes the programming languages dealing with images to which a meaning must be associated. The "e" symbol denotes empty, the " ' " symbol denotes an obtained description (logical or graphical).

More formally, an icon G may be defined as a quintuple (like in Automata Theory),

G (VL, VP, S, $x_\circ$, R) where

VL is a finite non-empty set of logical objects
VP is a finite non-empty set of physical objects
S is a finite non-empty set of icon names
$x_\circ$ is an element of S which denotes the head icon name
R is a mapping of S in $2^{VL \cup S \times VP}$

Rules R specify the icons as dual representations: a set of logical objects associated to a physical object. After the formal definition of an icon we may also define an icon algebra or a set of icon operations which allow combinations (vertical, horizontal) or icon enhancement by adding attributes of icon B (which is similar to icon A) to the logical description of icon A, sketching (which simplifies a logical description of an icon reducing it to its essential features), cross-indexing (finding icons which share some property in their logical description), etc. A great number of these operators which relate logical/physical descriptions between different icons may be defined according to the needs.

Moreover icons may not only be passive (objects) but they can also be active (processes) according to which modality has been triggered within the icon interpreter. In a printing program an icon representing an output unit like Thinkjet can be chosen (by means of one mouse button) or, if chosen by another mouse button, the same Thinkjet icon will invoke the driver program that will execute the printing of the selected documents. Before accepting the icon command a match will be performed between the process type and the icon type and, if compatible, the process will be stored in a multiprocess.list; this list is used to automatically create and update a function to be executed as the corresponding icon is selected.

Within this approach and in order to simplify Lisp programming and visibility an iconic language has been introduced (IconLisp[20]) where the icons are defined as triples (P, L, N) such that P is the physical component (containing a graphical representation which is a natural metaphor of the semantic value of the icon), L is the logical component (contains the Lisp list, equivalent to the standard listing of any program) and the name N (an identifier used for mnemonic and matching purposes).

Three classes of icons may be considered: object icons (constant or variable), transit icons and function icons. Bottomless wells contain an unlimited number of object icons (a label on the well will tell how many of these are active), programs or program excerpts will be contained in transit icons, their name being (by default) made by the first ten characters of the correspnding Lisp list (conversely they may be user-defined).

A workspace (the desk) and a palette of FUN-icons are available in this environment, the primitive and compiled functions are visually distinguishable since they are black on white background while the interpreted functions are white on black background. The desk may display three windows (PRETTY, ICONPRETTY and RESULT) which contain the logical component, its visualization and the last Lisp evaluated program output.

In this approach, as in all the interactive/visual environments, the differences between the interpreter, the editor (of syntactic type) and the system software

tend to disappear: we program, we correct, we input and visualize all processes including the output of our program within the same programming environment.

Within the new parallel image processing architectures (SIMD, multi-SIMD and MIMD) the problem of allowing every processor to execute its instruction concurrently with others is the main difficulty for the maximum usage of computing resources so as to obtain the greatest speed-up from the multiprocessor architecture. For a SIMD architecture the problem is relatively straightforward since only one kind of instruction is executed at once and therefore if the number of image pixels equals the number of processing elements there is no overhead in processor management.

For MIMD or even for multi-SIMD architectures the problems of task distribution and synchronization and their visualization may be of some relevance. Recently[21] a new graphical language (Pigsty) for concurrent programming has been suggested based on Pascal and Communicating Sequential Processe. This language (and its environment I-PIGS) allow the construction and visualization of a concurrent program animating the data communication between the processes (in a special window). Other possibilities include the display of the structured chart of control constructs (made of boxes containing a process, a procedure, a construct, etc) of a process and the tables of variables used in a process.

A program may be viewed as an abstract object implementing an algorithm, generally in textual form. In the case of images both the data structure and the multiprocessor architecture are n-dimensional (typically n=2,3...) and in the case of a pyramid architecture (such as the ones discussed in this Workshop) the base of such a pyramid is generally matched to the input image to be processed. The visual approach stresses the importance of using a graphical representation of the program where the different units (sets of program constructs representing processes by box-like icons) may be associated to other boxes so describing the communicating paths. The Pigsty language is based on Pascal for the sequential part-which corresponds to the succession of processes along the vertical direction-(from the base to the top of the pyramid in a bottom-up approach or from the vertex or a root node downwards in a top-down approach) and on Communicating Sequential Processes for the description of concurrency.

The same representation holds for the EXIT-LOOP construct or for the RETURN construct, WHILE construct, etc. I-PIGS provides editing functions to create single boxes and arrays of boxes (the pyramid architecture could have pre-defined base processors and i-level processors which may run processes, their connections may be activated (or inhibited) on the horizontal plane (to the four brothers) or vertically (among planes, to father and children).

An array S of four processes may be declared in CSP as follows:

"S(i:1..4) :: ...coding for the process"

at this stage the I-PIGS environment will display a row (or a square) of boxes that will represent the processes. Ports will be used for communication: InPort and Outport for receiving and sending data, the editor will guarantee correct communication paths (four-connectivity on the plane and five-connectivity between planes for a pyramidal architecture). Type checking of the data is performed at run time since Pigsty may broadcast any data type, in image processing a binary data type is often considered specially when using large quantities of elementary processors which are single-bit processors.

The communication and synchronization between the processes is performed whenever both the OutPort and the InPort are ready to send and receive, otherwise the process will have to wait, port names are used since ports are active and, for example, an output command is

OutPort '!' expression

while the input command is

InPort '?' variable

Adding and deleting links (under the editor) establishes communication between processes (the box-icons), both bottom-up and top-down development (abstraction and refinement) as required in the most interesting pyramidal algorithms (see the papers in[22]). The editing functions of I-PIGS allow the box decomposition (refinement) into a subsystem of boxes and a grouping of a system of boxes into one single subsystem (abstraction). Another interesting editing command allows the possibility of linking an array of processes (a pyramid plane) to another array of processes (the neighboring pyramid plane) by choosing corresponding pairs of InPort and OutPort. In the pyramid case, for two planes i, and i+1, the top plane will have a smaller number of processors (2^2 times less) so that the user will decide how to group the OutPorts to the InPorts, the interconnection between the ports is semantically important while the graphical editor of I-PIGS guarantees that a syntactically correct Pigsty process structure is built.

The possibility of using this language (and its environment) for programming algorithms on pyramidal architecture is based on its capability of representing both concurrent processes diagrammatically and the control structure of logic flow as well as the editing facilities which ensure correct syntax. Moreover, the display of the data communication, logic flow and updating of variables are helpful in analyzing the program and allows to follow it step by step.

Conclusions

The international debate on the ways in which programming languages may evolve and be improved (to make a better use of human resources and abilities) also bears on the image processing community. The problem is even more delicate since new architectures using large amounts of processors are being built and must be programmed easily and safely. From the old Fortran programs to the recent visual languages two interesting milestones are the extension of high level structured languages and the concept of an integrated language able to cope with all the functions required in an image information system. Perhaps new languages for programming multicomputer systems, such as pyramidal architectures, may use recent ideas for concurrent programming with visual languages.

References

1. K. Preston, Jr., "Progress in image processing languages", in Computing Structures for Image Processing, edit. M. J .B. Duff, Academic Press, London, 1983, pp. 195-211.
2. E. G. Johnston, "The PAX II Picture Processing System", in Picture Processing and Psychopictorics, edits. B. Sacks Lipkin and A. Rosenfeld, Academic Press, New York, 1970, pp. 427-512.
3. M. J. B. Duff and S. Levialdi, edits., Languages and Architectures for Image Processing, Academic Press, London, 1981.
4. S. Levialdi, A. Maggiolo-Schettini, M. Napoli, G. Tortora, G. Uccella, "On the design and implementation of PIXAL, a language for image processing", in 3., pp. 89-98.
5. A. P. Reeves, "Parallel Pascal: An extended Pascal for Parallel Computers", Journal of Parallel and Distributed Computing, Vol. 1, n° 1, 1984, pp. 64-80.
6. K. E. Batcher, "Design of a Massively Parallel Processor", IEEE Trans. on Computers, Vol C-29, N° 9, , 1980, pp. 836-840.
7 D. H. Lawrie, T. Layman, D. Baer, J. M. Randal, "GLYPNIR: a programming language for ILLIAC IV", Com. ACM, vol. 18, 3, 1975.
8. M. Zloof, "Classification of Visual Programming Languages", Proc. of IEEE Workshop on Visual Languages, Hiroshima, 1984, pp. 232-235.
9. B. D. Shriver, "Software paradigms", IEEE Software, Jan. 1986, vol. 3, n° 1, pp. 2.
10. B. Hailpern, "Multiparadigm Languages and Environments", IEEE Software, Jan. 1986, vol. 3, n° 1, pp. 6-9.
11. S. Levialdi, "Languages for Image Processing", in Fundamentals in Computer Vision, edit. O. Faugeras, Cambridge University Press, 1983, pp. 459-478.
12. Spider, Subroutine Package for Image Data Enhancement and Recognition, JSD, Yuseigojyokai-Kotohira Bldg. 14-1, 1-chome,

Toranomon, Minato-ku, tokyo, Japan.
13. S. Levialdi, A. Maggiolo-Schettini, M. Napoli, G. Uccella, "PIXAL: a high level language for image processing", in Real Time/Parallel Computing, M. Onoe, K. Preston, Jr., A. Rosenfeld, edits., Plenum Press, 1981, pp.131-143.
14. A. P. Reeves, "Languages for Parallel Processing" in Proc. of IInd Int. Workshop on Data Analysis in Astronomy, Erice, P. Crane, V. Di Gesù, S. Levialdi, L. Scarsi, edits., Plenum Press, 1986 (in press).
15. S. K. Chang, E. Jungert, S. Levialdi, G. Tortora, T. Ichikawa, "An Image Processing Language with Icon-Assisted Navigation", IEEE Trans. on Software Engineering, Vol. SE-11, N°8, 1985, pp. 811-819.
16. DBTG Rep., CODASYL Data Base Task Group, April 1971.
17. S. L. Tanimoto, E. P. Glinert, "Designing Iconic Programming Systems: Representation and Learnability", TR-85-07-05, Dep. of Computer Science, University of Washington, Seattle, 1985.
18. T. Ichikawa, S.K. Chang, edits. IEEE Workshop on Visual Languages, 612, ISBN 0-8186-0612-6, Hiroshima, 1984.
19. S. K. Chang, "Icon Semantics - A Formal Approach to Icon System Design", IPL-85-08-US-05.
20. G. Cattaneo, A. Guercio, S. Levialdi, G. Tortora, "IconLisp: an example of a visual programming language", IEEE Workshop on Visual Languages, Dallas, 1986.
21. Man-Chi Pong, "A Graphical Language for Concurrent Programming", IEEE Workshop on Visual Languages, Dallas, 1986.
22. A. Rosenfeld, edit. Multiresolution Image Processing and Analysis, Springer-Verlag, Berlin, 1984.

A HIGH LEVEL LANGUAGE FOR PYRAMIDAL ARCHITECTURES

V. Di Gesu'

Dip. di Matematica ed Applicazioni, Univ. of Palermo
Via Archirafi 34, 90100 Palermo, Italy.

I.F.C.A.I./C.N.R.
Via Mariano Stabile 172, 90100 Palermo, Italy.

ABSTRACT

In the paper are described the syntax and some implementation features of a high level language for pyramidal architectures called Pyramid C Language (PCL). The language is an extension of the C and include data type, set of instructions and builtin functions oriented to the pyramidal architectures. Some notes on the implementation for the PAPIA machine are also given.

Key words: Languages for I.P., Image Analysis, Pyramids.

1. INTRODUCTION

Up to now several pictorial languages have been developed and proposed in order to handle high level vision problems [1,2,3,4]. They allow the definition and the manipulation of visual concepts and icon-guided navigation in the pictorial data base.

On the other hand there exist needs for design high level languages in order to write algorithms for the solution of "low level vision" problems on parallel machines. Parallel Pascal for the MPP [5] and the C-dialect designed for the CLIP-series machines [6] are examples of such languages.

A good overview on pictorial languages is given by K.Preston Jr. in [7]. In the paper the author gives a general description of a wide class of existing systems for image processing.

In the following are described the main features of an high level language for pyramidal architecture, called "Pyramid C Language" (PCL).

PCL is an extension of the C language, which includes both serial and parallel data-type, parallel instructions-set. Builtin functions are, also, provided to allow easy navigation in the pyramid architecture (shift, propag, rotate, path,...) and to compute "low level vision" functions (edge detection, contours, filtering, Hough-transform, statistics...).

NATO ASI Series, Vol. F 25
Pyramidal Systems for Computer Vision
Edited by V. Cantoni and S. Levialdi
© Springer-Verlag Berlin Heidelberg 1986

A C-extended language is proposed because C has a complete and elegant syntax, that allows to write well structured programs. It provides versatile interlanguage-interfaces in order to built programs with several sources written in different languages. Furthermore there exist a trend toward the use of C as a standard under the UNIX operating system, which imply its portability among heterogeneous hardware.

In the design of the PCL-compiler a multilayer strategy has been adopted in order to achieve an user vision nearly hardware-independent.

Section 2 is devoted to the description of the PCL constructs and data type. Section 3 outlines the strategy adopted to built the compiler for the PAPIA machine [8]. In Section 4 are given concluding remarks.

2. THE PCL LANGUAGE

In the following are described some parallel features of the PCL. Data and instructions are addressed to a virtual pyramid machine, where processor elements (PEs) have their hown ALUs, registers and memory units. The PEs are arranged in a quadtree and are four/eigth connected at each plane level.

Separation is made between serial and parallel variables or costants and serial and parallel statements. This feature allows a better conceptual separation between serial and parallel components of the algorithms. Serial and parallel sections of the algorithms are executed concurrently in order to have run time performance.

Parallel Data Types.

Parallel data types allows to represent basic pictorial objects. At the present "images" and "masks" in each plane of the pyramid and the "pyramids" as hierarchical quadtree are provided. The "pyramids" represent also multiresolution images. The syntax is:

```
<parallel-type> ::= ima of <simple-type> <ident>[<size>]/
                    pyram of <simple-type> <ident>[<height>]/
                    mask <ident>[<size>].
```

Where <size> defines the linear-size of both images and masks, <height> indicates the number of the pyramid-levels and sets automatically the linear-size of its base to 2**(<height>-1). At the present only squared images, masks and pyramids are provided.

Parallel handling of array data types is foreseen, its efficiency depends from the hardware architecture. However algorithms for "low level vision" do not require often the use of array indexing.

EXAMPLES. Suppose that the following declarations have been made:

```
pyra of int P[3];
ima of int M[4], N[4];
int K;
```

than P[3] defines the first level of P and P[3] = M assigns M to the base of P, whereas P[2] = M will generate size-error.

The <simple-type> supported by PCL are character, integer, short integer, floating, and boolean. The boolean type has been inserted because is extensively used for binary images. Masks are boolean by default.

Parallel Instructions.

Parallel instructions are addressed to the pyramid PEs, they are classified in five main groups:

- set-context statements;
- assignement statements;
- control and iterative statements;
- selection statements;
- I/O statements;

Set-context statement.

This statement has been introduced in order to define the context where the parallel variables are defined and the computation is performed. Its formal definition is:

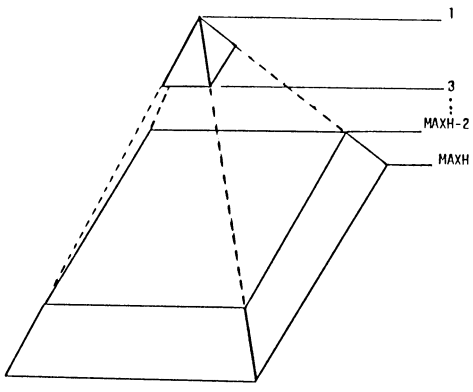
```
set context {<apex>},{<height>};
```

Where <apex> and <height> are integer. The value <apex> defines the top plane whereas the <height> defines the number of working planes. By default <apex> is set to 0 and <height> to the maximum heigth (MAXH) of the pyramid (bottom level). MAXH depends from the hardware complexity and its value is generated at compile time. As an example the statements:

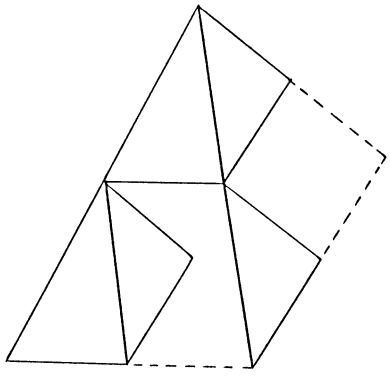
```
set context *,3;
set context MAXH-2,2;
```

define, as working area, a pyramid with heigth 3 starting from the top plane, and last two planes of the pyramid-machine (see Fig. 1a)

Set context is a very useful statement that allows the user to handle in easier way the pointers to the PEs by means of automatic entry-point. Set context may be used in order to run algorithms on separate section of the pyramid hardware. Future developments will allow to define vertical slices (trees) and disjoint subpyramids (see Fig. 1b).



(a)



(b)

Fig. 1

Assignement statement.

Its syntax is equal to the corresponding serial one. The assignement statement is defined formally as:

<assignement> ::= <ident> <aop> <expression>;

The assignement operator <aop> may be parallel or sequential depending from the type of <ident> or <expression>. The assignment operators are:

= , != , ~=

where '=' is the standard assignement, '!=' stays for assign the logical negation and '~=' stays for assign the one's complement.

For the variables before declared the following parallel assignements hold:

```
M = N;
M = K;
```

Control and iteration statements.

The PCL provides all the control statements of C, in the following are described only the parallel version.

The 'where-otherwise' controls the flow of the program in each PE, it operate as a parallel if-else statement. Formally it is defined:

```
where (<expression>) <statement>
otherwise <statement>;
```

This statement is realized in two phases by the controller firmware of the pyramid, by setting "on" the mask-registers of the PEs.

The 'pwhile' and 'do-puntil' statements operate as a parallel while and do-until in each PE, their sintax is:

```
pwhile (<expression>) <statement>;
do <statement> puntil (<expression>);
```

Termination conditions are detected by the microcoded control unit of the pyramid by means of the OR-SUM test.

Selection statements.

The selection statement defines a subset of pixels on which the computation is performed. Selection on an image is realized by means of a "mask" variable or expression, that plays the same role of a characteristic function for a set. The syntax is:

```
<ident>(<mask-exp>)
```

Where <ident> is of type "ima" and <mask-exp> is an expression of masks. For example, given the masks m1 and m2 it is possible to select the pixels of the image M as follows:

```
M(m1) = K;
M(~m1) = K;
M(m1 ∧ m2) = K;
M(m1 ∨ m2) = K;
```

The selection on pyramid is performed by indexing the layer and by masking the layer chosen. The syntax is:

```
<ident>[<index>](<mask-exp>)
```

As an example the statement:

```
P[3](m1) = M(m2);
```

assigns the pixels of the layer 3 of P selected by m1 to the corresponding value of M selected by m2.

Extended selection is provided in order to point classes of pixels or a single pixels. In this case has been used the array notation of the C language. For example:

```
M[*][1] = N[*][2];
```

assigns the second colum of N to the first one of M.

Extended selections provides also to assign the 'odd/even' columns of N to the 'odd/even' columns of M:

```
M[*][odd] = N[*][even];
```

The statement:

```
where (P[3][*][odd] > 2)
      M[*][odd] = 1;
otherwise
      M[*][odd] = 0;
```

performs the tresholding of M under a condiction statement computed on the odd columns of the bottom plane of P.

I/O statements.

I/O functions dedicated for the transfer of the image-files (pyramid-files) from the host-computer to the PAPIA processor and viceversa are provided through the active memory.

Examples are:

```
LOAD(M,<from-file>,<to-plane>);
UNLOAD(M,<from-plane>,<into-file>);
```

I/O of scalar values are also performed from and to the "apex" (READPEX, WRITEPEX).

Builtin functions.

PCL provides several builtin functions to perform standard computation tasks. They are part of a library and are directely callable from the user source program. They provide data distribution and navigation in the context, standard statistical and logical computations, neighbour-operations.

As an example of navigation functions SENDDOWN provides for the down propagation of the pixel-values of p1 starting from the plane p3 to the pixel-son, p4, belonging to p2. The syntax is:

```
senddown(p1,p2,p3,p4);
```

Where p1 is of type 'ima', 'mask' and 'pyra', p3 is of type integer and p4 may assume its values in the set {NW, NE, SW, SE}. For example:

```
senddown(M,P,3,NW);
```

The corresponding actions taken are:

- set in the context all the plane from 3 throught the bottom;
- assign the value of each pixel-value of M in the ith plane (3<i) to the Nord West sons of P.

Builtin functions dedicated to data distribution and navigation are:

ROTATE, SHIFT, SENDUP, SENDDOWN, PROPAG, DILATE, EROSE

Builtin functions for statistical and logical computation are:

MIN, MAX, SUM, AVERAGE, HAAR, HOUGH, ALL, ANY,...

Builtin functions for neigbour-operations are:

ORNEIGH, ANDNEIGH, XORNEIGH, SELNEIGH,...

Builtin functions may be combined to perform complex operations as:

```
SENDUP(MAX(M),P,3)
```

The maximum value of M will be sent to the respective father until the plane 3 is reached.

Example of PCL program.

In the following is given an example of PCL program to compute the discrete convolution of an image M, of linear size N, by means of the kernel K, of linear size D, the results are in the image M'. The algorithm is based on the formula:

$$M' = K * M$$

Where * is the convolution operator.
The identifier "image" and "kernel" are the input files containing the image to be deconvolved and the deconvolution kernel, while "decon" is the output file containing the deconvolved image.

The algorithm, written in PCL, is displayed in Fig. 2.

```
#define MAXH 7
main()
 {
  set context MAXH-2, 2;
  int D, i, j;
  ima of float M[128], M'[128], K[128];
  pyra of float P[2];
  LOAD(M, image, 2);
  LOAD(K, kernel, 2);
  SHIFT(SHIFT(M', D, N), D, W);
  SHIFT(SHIFT(M, D/2, N), D/2, W);
  for (i = D; i >1; i--);
   {
    P[0] = SUM( M*K );
    SENDDOWN(P,M',0,NW);
    SHIFT(M',1,E);
    SHIFT(M,1,E);
   }
   SHIFT(M',1,S);
   SHIFT(M,1,S);
   SHIFT(M',1,W);
   SHIFT(M,1,W);
  }
  UNLOAD(M',2,decon);
 }
```

Fig. 2

3. IMPLEMENTATION NOTE

The advent of new computer architectures dedicated to the image processing (I.P.) has introduced new ingredients in the definition of the programming languages:

LANGUAGES := INSTRUCTIONS+DATA-STRUCTURES+HARDWARE-STRUCTURES

In the design of new programming languages for I.P. (PLIP) running on parallel processors the above ingredients must be considered in order to achieve:

* INNER PARALLELISM
* EXECUTION EFFICIENCY
* SELF-EXPLICATION
* HANDLING OF THE CONCURRENCY

Nevertheless well designed PLIP are also requested to be:

* PORTABLES
* GENERAL PURPOSE

The last two features may conflict with the first ones because of the hardware dependence. Multilayer compiler implementation strategy may solve partially the problem, by relegating the hardware dependence to the lower levels. Such that each level corresponds to a virtual machine which is

connected to the layer application. Fig. 3 shows such view in the framework of the pyramidal architectures. The "//" indicates the interface between two adiacent levels.

```
HIGH LEVEL VISION <-----> PICTORIAL    LANGUAGES
--------------------//--------------------------
LOW LEVEL VISION  <-----> HIGH  LEVEL  LANGUAGES
--------------------//--------------------------
CONTROL FUNCTION  <-----> CONTROLLER   ASSEMBLER
--------------------//--------------------------
PYRAMID  FUNCTION <-----> PYRAMID MICROASSEMBLER
```

Fig. 3

In the design of the compiler for the PAPIA machine a two-stages architecture has been adopted.

The first stage consists in a precompiler which taking as input the PCL source code returns a PMACRO code version [8,9]. It allows to allocate all the variables, to recognise their type, subrange and to classify them as serial or parallel. It provide also the translation of the parallel components of the PCL source in the corresponding macro instructions with appropriate parameters.

The second stage is the preprocessor of the PMACRO, which allows to separate the serial and parallel components of the algorithm and produce the executable code for the Microprogramed Control Unit (MCU) [10].

PMACRO is a C-like metalanguage in which macro instructions for the pyramid hardware and the control unit are interleaved with standard C. The parallel sections of the algorithm are recognized from the serial-ones by means of a "%" symbol in column one. In Fig. 4 is given a sample of program written in PMACRO.

```
/* program to compute the sum of two integers
   loaded in the shift registers  sr1  andsr2
  of each PEs, the  result in  sr2 is  stored
  in the local memory */
#define PRECISION 8
main()
 {
  int i;
% SET C
  for (i = 1; i<=PRECISION; i++);
   {
%   SBS sr1,sr2
   }
  for (i = 1; i<=PRECISION; i++);
   {
%   ST2 200+i
   }
 }
```

Fig. 4

Some limitations of the PAPIA hardware configuration could reduce the number of variables and the precision, however such limitations do not alter the generality of its definition because are foreseen parametric at compilation level.

Several processes may be runned at the same time in different sections of the pyramid hardware. In this case the language provides macro-instructions, that address each process (program) of the task to the appropiate plane of processors. At the present PAPIA will operate in SIMD mode and PCL will not include multy-task and concurrency handling.

In its first release PCL will be compiled in a host computer provided of a C-compiler generating an object code for the Motorola 68000 microcomputer used in the front-end electronic of the pyramid controller.

The precompiler, written in C-language under UNIX operating system, is in progress stage. The preprocessor of the PMACRO, written also in C-language under UNIX operating system has been completed and PAPIA users may write their algorithms in PMACRO.

4. CONCLUDING REMARKS

The PCL design is at a preliminary stage, users may write algorithms in PMACRO. PCL has been oriented toward the computation of "low level vision" function, therefore more global and local image functions have been developed than array indexing manipulation.

Future developements will include handling of concurrent processes and the definitions of more general data-type (binary-tree, quadtree,...) and more powerful context statement.

PCL has been designed for a class of not-interleaved pyramid-machines with static links between the PEs. Nevertheless its definition could be extended to a wide class of pyramid-machines.

ACKNOWLEDGEMENTS. I would like to thank all the PAPIA-staff working in both the Palermo University and the IFCAI/CNR, Prof. S. Levialdi for the useful suggestions, Prof. G. Gerardi and Dr. A. Machi' for the useful discussions on some of the concepts presented in the present paper.

REFERENCES

[1] S. Levialdi, A. Maggiolo Schettini, M. Napoli, G. Tortora, G. Uccello, "On design and implementation of PIXAL, a language for image processing", in Language for Image Processing, edits. M. J. B. Duff and S. Levialdi, Academic Press, London, 1981.

[2] G. Cattaneo, S. Levialdi, G. Tortora, "Language for Image Manipulation", IEEE Workshop on Visual Languages, Hiroshima, pp. 108-114, 1984.

[3] S. K. Chang, E. Jungert, S. Levialdi, G. Tortora, T. Ichikawa, "An Image Processing Language with Icon Assisted Navigation", IEEE Trans. on Soft. Engineering, Vol. SE11, pp. 811-819, August 1985.

[4] G. Cattaneo, A. Guercio, S. Levialdi, G. Tortora, "IconLisp: an example of visual programming language", IEEE Workshop on Visual Languages, Dallas, June, 1986.

[5] A. P. Reeves, "Parallel PASCAL: An extended PASCAL for parallel computers", J. of Parallel and Distributed Computing, Vol. 1, pp. 64-80, 1984.

[6] D. E. Reynolds, G. P. Otto, "Software tools for CLIP4", Report No. 82/1.

[7] K. Preston Jr., "Progress in Image processing", in Computing Structures for Image Processing, Academic Press, London, 1983.

[8] V. Cantoni, L. Carrioli, O. Catalano, L. Cinque, M. Ferretti, G. Gerardi, S. Levialdi, R. Lombardi, A. Machi', R. Stafanelli: "The PAPIA image analysis system", SPIE International Symposium on Image Processing, Cannes, 1985.

[9] O. Catalano, G. Di Gaetano, V. Di Gesu', G. Gerardi, A. Machi', D. Tegolo, "Low level languages for the PAPIA machine", Proc. of the II Int. Workshop on Data Analysis in Astronomy, Plenum Press, Erice, in press.

[10] G. Gerardi, "Hardware for PAPIA controller", presented at NATO ARW on Pyramidal Systems for Image Analysis and Computer Vision, Maratea, 1986.

TACTILE INFORMATION PROCESSING*

R. Bajcsy
S. Stansfield

GRASPLab, University of Pennsylvania
Philadelphia, Pa. 19103

ABSTRACT

In this chapter we present an introduction to tactile information processing. We begin by describing the technology and presenting some early work; we then discuss in detail our current work in building a robot haptics system; finally, we introduce the concept of shape from touch and briefly describe two systems which incorporate touch with vision to do object recognition.

1. Introduction

We define tactile information processing, or tactile perception, to be a combination of both cutaneous data (ie. the response of the sensor to contact) and kinesthetic data (ie. the position and movement of the sensor). At the University of Pennsylvania's GraspLab, we have been involved with research into tactile processing for several years, beginning with our early work in sensor testing and calibration[1] and continuing through to our most recent work in the development of robotic tactile systems[2, 3].

We would like to accomplish two tasks during the course of this chapter. The first is to introduce, if only briefly, several of the stages of tactile processing. Toward this end, we will spend the first part of the chapter discussing tactile sensing devices and the testing/calibration techniques we have applied to those sensors with which we have worked. Our second goal is to describe our most current research into the development of an independent tactile perception system. Therefore, we will spend the bulk of the remainder of the chapter describing our ongoing work with the ARAMIS[3] system -- its design, control,

*This work was supported by the following grants: ARO DAA6-29-84-k-0061; AfOSR 82-NM-299; NSF MCS-8219196-CER; NSF MCS 82-07294; AVRO DAABO7-84-K-FO77; NIH 1-RO1-HL-29985-01.

NATO ASI Series, Vol. F 25
Pyramidal Systems for Computer Vision
Edited by V. Cantoni and S. Levialdi
© Springer-Verlag Berlin Heidelberg 1986

and representations. The ultimate goal of the ARAMIS system is to combine its tactile perceptual system with a vision system under the control of a set of knowledge-based experts for the task of object categorization. We will discuss this future work briefly in section six of the chapter. Finally, we wish to point out that an excellent previous work into the development of a bimodal (vision/touch) system was that of Allen[2]. Because the high level goals of this system and that of ARAMIS are similar, we will not discuss this work in detail. However, many of the methodologies used by Dr. Allen differ from those of the current research; and so we will include, as the last section, a brief discussion of this system in order to show an alternate solution to the problem.

2. Sensing Devices

One of the greatest handicaps to research in tactile information processing has been the lack of accurate and reliable sensing devices.

Bajcsy[1] lists several properties which are desirable for a tactile sensor. Among them are

- spatial resolution < 2 mm
- deformable surface
- reaction time < .1 sec
- no hysteresis
- realiability and uniformity
- ruggedness

Most sensors today are based on one or more of the following technologies[1].

- strain gauges
- microswitches
- pieroelectric changes
- magnetistriction
- change of resistence with compression
- piezoresistence

These sensors may be used in several different scenarios. With a single finger/sensor, we may obtain information about contact/non-contact and three dimensional position; if our sensor is capable of greyscale measurements, we may determine local

characteristics of the object touched, such as texture. We may also "poke" at the object to obtain elasticity and compliance information. We can follow edges and surfaces. And finally, we may obtain kinesthetic information such as surface normals and curvature, depending upon the capabilities of our device. With two and three fingered scenarios, we may obtain all of the above information as well as spatial extent, rough shape, and mass by grasping the object and hefting it.

All of the experiments described in this chapter use the one finger scenario. The two primary devices used have been the LAAS sensor (from LAAS , Toulouse, France), and the LORD LTS-200 (from LORD Corp., Cary, North Carolina.)

The first, used in Allen's work, is about five inches long, with an octoganal cross section about 3/4 inches in diameter. Each of the eight rectangular faces is connected to a tapered piece, which is in turn connected to a common tip piece. There are a total of 133 sensitive sites -- 16 on each main face, one on each alternate taper, and one on the tip.

The second, used in the ARAMIS system, is a combination tactile array and gross load sensor. The array sensor is composed of sensitive sites imbedded in an elastomeric touch surface. There are 160 sites arranged in a 10 X 16 orthoganal array. The gross load, or vector, sensor measures forces and moments along and about the three coordinate axes at the touch surface. The sensor consists of a Maltese Cross arrangement of four beams with a set of strain gauges on either side of each beam (see Buchner[4].)

Both the LAAS and the LORD LTS-200 have been mounted on the wrist of a PUMA 560 robot arm and interfaced to the Unimate system via a z80 Single Card Computer. Each is also interfaced to a VAX 750, and it is the network of these computers which comprises the hardware for each of the tactile systems.

3. Calibration and Testing

3.1. Early Work with the LAAS

An important first stage in tactile processing, particularly with experimental as opposed to commercial devices, is to understand the physical properties of the sensor. In this way, we may calibrate the device for future use. Early experiments with the LAAS sensor were designed to do exactly this. These experiments are described in Bajcsy[1]. We discuss them briefly here to provide an indication of the type of testing which may be done.

3.1.1. Determining the Physical Properties of the Sensor.

The uniformity and time response of the LAAS sensor were measured by applying a uniform weight to each sensor site and then taking readings at fixed time intervals. In this way, one may determine how long it takes after contact for the readings to "settle." This is especially important in cases where the tactile image will be used in non-static mode (static mode is also referred to as snapshot mode.) Repeatability is determined by performing the same experiments at different times and then plotting the differences in readings. Hysteresis is determined by plotting the readings as the sensor moves into and away from a suface. The measure of hysteresis is the area under this curve.

Finally, the hardness of a material can be characterized by the slope of the linear portion of the above curve as the sensor moves into and away from the material. The larger the slope, the harder the material. Table 3-1 shows the hardness measurements for several materials tested in this way.

material	slope toward	slope away
wood	1.69	2.7
felt	1.08	1.2
foam	1.18	2.3

Table 3-1: Hardness Measurements Obtained with the LAAS[1]

3.1.2. Texture

Texture experiments with the LAAS were performed in a dynamic mode: The sensor was contacting the surface and moving translationally. Several textures of varying fineness were tested by moving the finger over the surface and recording a reading every 5.6 ms. The sensor was able to distinguish a very course surface (wire grid) from a smooth surface (felt). Again, these experiments are described, and their results presented, in Bajcsy[1]. The

reader is referred to this paper for further details.

3.2. The LTS-200

The array component of the LTS-200 gives us a 10 X 16, 15 level greyscale tactile image; and even in static sensory mode, this allows us to do what we will refer to as automatic calibration: During the performance of an experiment, as a tactile image is obtained, statistics (average and standard deviation of greyscale values) are calculated. These statistics are then used to determine the characteristics of the touched surface, that is, to decide whether the sensor is flush on a surface, or only in partial contact. Based upon which of these situations is true, the statistics are again used to either characterize the roughness, or texture, of the surface (if the sensor is flush) or to automatically generate a threshold for future processing (if it is not.) It is the determination of this threshold to which we refer when we speak of automatic calibration. Hence in one process we may calibrate and characterize the texture of a surface.

Table 3-2 shows the results of experiments performed using this technique. Each of the fourteen objects listed in the table was approached from above until contact with the sensor was made with a component force in any direction of greater than or equal to 0.5 lb. The deflection array was then read and the mean and standard deviation of the deflection values over the face of the sensor were calculated. These values are also listed in Table 3-2. We determine if the sensor is resting flush upon a surface larger than the sensor face by looking at these statistics. If the standard deviation of the grey values is larger than the mean, then either we are in contact with an object smaller than the sensor face (nut and screw tip are examples); or we are in only partial contact with a larger object, due either to the positioning of the sensor (corner and edge are examples), or to the geometry of the object itself (cylinder and bowl rim are examples). If the standard deviation is smaller than the mean, then we are flush on a surface and contacting over a large portion of the sensor face. In this case, we can use the statistics to decide whether we are on a smooth or textured surface -- a small standard deviation compared to the mean (say less than half) indicates a smooth surface (table top, plate, and pad are examples). A large standard deviation indicates a textured surface (rice, magnolia seeds, roughened styrofoam, and woven cloth are examples). Figure 3-1 shows the thresholded deflection grey values for

magnolia seeds. We can get some idea of the "roughness" of the texture from the standard deviation as well. The larger the deviation, the rougher the texture. Thus rice is smoother than styrofoam, which is in turn smoother than magnolia seeds.

object	mean	standard deviation
corner	3.84	4.31
cylinder	4.58	5.81
edge	7.85	10.22
grid	11.61	5.63
magnolia seeds	8.75	8.01
nut	6.34	7.70
pad	11.80	3.36
plate	10.95	3.76
rice	7.70	3.62
bowl rim	3.63	4.28
screw top	2.06	5.21
styrofoam	8.40	7.84
table top	7.58	1.68
woven cloth	5.34	3.62

Table 3-2: Deflection Grey Value Statistics

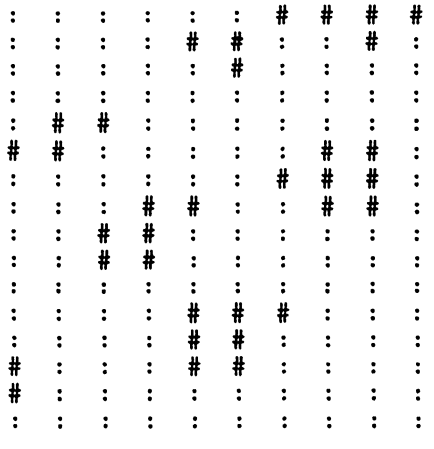


Figure 3-1: Seed Texture

4. Tactile Perception

We now proceed to the next stage of tactile processing -- that of building a tactile perception system. From this point on, we will be discussing the design, experiments, and results obtained from our current, ongoing research with the ARAMIS system, which uses the LORD LTS-200 sensor mounted on the PUMA 560 robot arm. In designing a robot

haptic system, we have looked both to the vision researchers and to the psychologists. Borrowing from methodologies and theories in both of these areas, we have divided our design into three major components: tactile primitives, tactile features, and the active exploratory procedures** used to locate and identify them.

4.1. Tactile Primitives

Klatzky and Lederman[5, 6] suggest that the haptic system computes information related to an object's form, substance, and function. Form includes measurements of shape and size, while substance represents the properties of an object such as compliance and temperature. We accept this model and furthermore require that a tactile primitive be directly measurable/computable (as opposed to derived) -- if a problem solving capacity is required to identify an entity, then it is a feature rather than a primitive. Thus, for example, temperature is a haptic primitive, while contour is a haptic (as well as visual) feature.

Given these conditions, we propose the following ten tactile primitives: compliance, elasticity, surface normal, point contact, area contact, edge contact, texture, mass, size, and temperature. Of the ten, we are not able to implement mass, size, or temperature given the devices available to us (mass and size require at least a two-fingered gripper; temperature requires a temperature-sensing device.) In addition, we have discussed texture in the previous section on calibration. Below we discuss the remaining six, along with the procedures used to extract them, and provide some preliminary results.

4.1.1. Compliance

Since we are only able to measure gross forces and moments over the face of the sensor, our determination of compliance is rough. We use the following procedure:

```
procedure Compliance

  move until contact surface with
   force F in any component.
  store position P1.
  press against surface until force = 2*F.
  store position P2.
  compliance = DISTANCE(P1, P2)
```

**The term exploratory procedure was first introduced by Klatzky and Lederman.

```
end
```

The distance which the arm moves before obtaining a force of 2*F over the face of the sensor may be used as a rough estimate of the compliance of the surface: a "hard" surface will not show a perceptible change between the two points, while with a "soft" surface, the distance between the two points will be relatively large. Table 4-1 shows the results of running this procedure for several different objects.

Object	Compliance
bubble wrap	5.60
folded table cloth	15.06
note pad	1.86
plate	0.00
rice	5.87
sponge	6.59
table	0.38
tube of toothpaste	4.46
wallet	2.20

Table 4-1: Compliance

4.1.2. Elasticity

While compliance tells us in some sense how soft or hard a surface is, elasticity gives us some idea of its malleability. Again, our elasticity determination is rough:

```
procedure Elasticity

  move until contact surface with
   force F in any component.
  store postion P1.
  push against surface until force = 2*F.
  store position P2.
  dist1 = DISTANCE(P1, P2).
  back away from surface.
  move until contact surface with force F.
  store position P3.
  dist2 = DISTANCE(P3, P2)
  elasticity = dist2/dist1

end
```

If elasticity is one, then the surface has returned to its original shape. If it is zero, then the surface did not recover shape at all. Values between zero and one indicate an elasticity between the two extremes. (We will only attempt to measure elasticity for surfaces which

are compliant.) Table 4-2 shows the results of running this procedure on seven objects of differing elasticity.

object	elasticity
bag of rice	0.12
bubble wrap	0.90
folded tablecloth	0.60
rolled sock	0.76
sponge	1.06
table top	0.00
tube of toothpaste	0.14

Table 4-2: Elasticity

4.1.3. Surface Normal

A surface normal measured at a single point will give us a measure of surface orientation. Several normals sampled over a local area will allow us to determine surface shape. (We consider shape to be a feature rather than a primitive and discuss it more fully in the next subsection.)

4.1.4. Contact

This is the most elementary, and perhaps the most important, of the tactile primitives. We define "contact" simply as the presence or absence of tactile sensation at a specified point. For example, if someone were to move your finger to a particular location and then ask you if you felt anything, your answer would be yes or no. If yes, you might elaborate that you felt something sharp or something blunt. We define three types of contact: area contact, which you might feel if your finger were placed on a large, flat surface (relative to your fingertip); point contact, which would be felt if your finger were placed againt the tip of a pen, say; and edge contact, which is exactly as it sounds -- your finger placed on the edge of a table, for example.

The exploratory procedure for all three contact primitives is similar: the sensor is brought to a specified location and the deflection grey levels are obtained. A threshold is determined and the contact is parameterized by the mean and standard deviation of its greyscale, the centroid of contact, the percentage of the fingertip covered, the elongation of the region, and the bounding trapazoid.

Area Contact

We define an area contact to be a region of contact which covers a large portion of the sensor pad and which does not vary greatly in deflection values over its region. This type of contact will most often be obtained by placing the sensor on a relatively flat, large surface. The surface may or may not be textured.

Point Contact

We define a point contact to be a region of contact which covers a small portion of the sensor pad and which may vary greatly in deflection values over its region. The extent of the variation in grey values will give us some idea of the "sharpness" of the object. This type of contact will most often be obtained by placing the sensor on an object smaller than the sensor pad, or on some portion of a larger object which is either smaller than the sensor pad or which is very sharp, such as a corner.

Edge Contact

Our final primitive is the edge contact. We define an edge contact as a region of contact which is highly elongated and which is surrounded on either side by areas of non-contact. Figure 4-1 shows an edge contact obtained by exploring the boundary of a book.

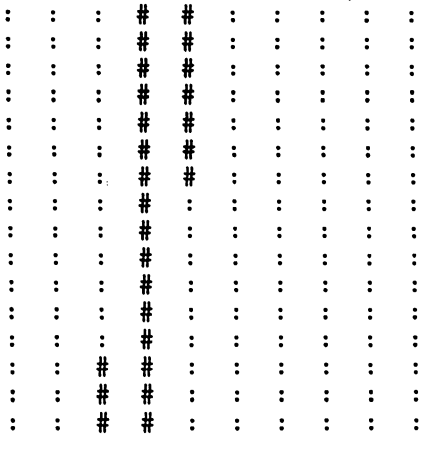


Figure 4-1: Edge Contact

4.2. Tactile Features

A feature is a derived entity. The identification of a feature requires some problem solving capabilities. A feature is made up of one or more primitives or features. We base our definition upon psychological theories (See Rock[7]) that the perceptual system, even at very early stages, is actually a collection of problem solving mechanisms. Hence features, which are the output of these mechanisms, are the result of a decision making process and therefore are not primitives. The rest of this section is dedicated to discussing tactile features and their associated exploratory procedures.

4.2.1. Edges

Our first tactile feature is the edge. An edge is a series of connected edge contacts. The exploratory procedure for detecting an edge is to follow it with our "finger" by determining the orientation and position of the edge contact we are currently "feeling" and using this to predict the postion and orientation of the next edge contact.

4.2.2. Corners

A corner is comprised of the intersection of two or more tactile edges of very different orientations and the tactile primitive point contact. Whenever an edge is terminated without being closed, there is a potential corner. If this edge is found to be connected to another edge at a nearly perpendicular intersection, then the sensor is centered on the point of intersection and rocked about the point until a sharp, small point contact is obtained. The presence of this primitive is the crucial factor: If it is not detected, then we do not have a corner.

4.2.3. Contours

A contour is a closed boundary made up of tactile edges. In the case of a smoothly varying contour such as the rim of a bowl, it will be a single edge which closes in upon itself. In the case of the boundary of an object such as a book, it will be a series of edges of different orientations and/or positions; in this case, the contour will also very likely contain corners.

4.2.4. Surface Patches

A surface patch is detected by the exploratory motion of surface following: an area contact or a large, blunt point contact indicates a surface patch. The surface patch is then explored locally by moving the sensor about the intitial contact position. The characteristics of the surface should not vary greatly from contact to contact. A surface patch is comprised of elasticity and compliance measurements; a texture characterization; and a set of normals. Three or more normals will allow us to categorize the surface as planar, cylindrical, or spherical.

4.2.5. Holes and Cavities

A hole is defined as a region of non-contact completely surrounded by areas of contact. The hole feature will be most valuable in distinguishing between small objects of which it is a feature. In addition, because of the geometry of our sensor, we will not be able to distinguish small cavities (ie those which we are unable to probe) from holes; we will classify these components as either holes or cavities. A cavity is a tactile contour surrounding an area of non-contact. It must be associated with a surrounding surface of which the contour forms a boundary. We may distinguish large cavities from holes by probing into the suspected hole/cavity with the sensor.

5. Shape From Touch

By this time, it is no doubt evident that the exploratory procedures used to locate the tactile primitives and features are purpositive, or goal-driven. We must, therefore, have a mechanism for invoking these procedures and for combining the sparse, disparate data which each gives us. We suggest that the heart of this mechanism lies in the representation used for the objects in our world.

The representation we propose is a hierarchy of frames. At the lowest level of this hierarchy are the frames which represent the tactile primitives; for example, the frame for a contact primitive would contain slots for the parameters which define the region and its 3-D world position. At the intermediate levels of the hierarchy are the tactile features which these primitives define. For example, the frame for a tactile contour would contain the frames for the tactile edges which comprise it. These edges would themselves be made up

of primitive edge contacts. At the highest level of the hierarchy, that of the object, the frame would contain slots for all of the features which define the object.

Thus the hierarchical frame representation provides the mechanism for triggering the exploratory procedures. These procedures are simply attached to the frame representing the features they are designed to extract. In the top-down mode of exploration, the object is verified by filling in each of its composite slots. The slots are filled by executing the associated exploratory procedure.

In either a bottom-up or a mixed-mode of operation, triggering of the tactile exploratory procedures will be guided initially by vision. When tactile exploration and verification of all visual portions of an object has been accomplished, hypotheses concerning the identity of the object may be generated and exploration will then proceed in a top-down fashion.

6. Future Work: Object Categorization

The methodology alluded to in the previous section obviously requires more sophisticated techniques than we have presented thus far. Future work on the ARAMIS system involves the integration of the tactile system with a simple vision system under the control of a set of knowledge-based experts. The ultimate goal is object categorization, or the recognition of objects at a basic, or generic level. To do this will require not only adequate tactile and visual information processing, but also an "intelligent" control system.

Object representation within the system will, again, be feature-based. (Features will be both visual and tactile and will include surface shape, dimensionality, cavities, and contours among others.) Objects will be represented as a hierarchy of frames: The highest level will contain a description of the whole object in terms of the features which comprise it and the expected relations among these features. At the next level will reside the representations of the individual features. These representations will contain information pertinent to the specific feature and will include slots for visual, tactile, and amodal properties of the feature. At the level below this will be the perceptual primitives (visual and tactile).

The architecture we propose is a distributed hierarchy of knowledge-based experts;

each expert will be both domain specific and informationally encapsulated. We base our design on current theories of the human perceptual system[8]. At the lowest level of this hierarchy are the sensors. At the next level reside the modules whose task is to perform low level processing of the raw sensory data in order to extract the primitives. Intermediate levels of the hierarchy contain the modules responsible for extracting and interpreting features. These modules are responsible for filling in the slots of the features frames in the previously discussed representation. There will be one module for each feature. Integration of the sensory modalities will also occurr at this level. Finally, at the highest level, are the strategists. These modules are responsible for the overall control within the system. They include the reasoner -- responsible for hypothesis generation, disambiguation, and verification; and the explorer -- responsible for generating sensing strategies once a goal (such as "determine the presence of feature X") has been formulated by the reasoner. Knowledge representation within the system will vary from module to module: The representation scheme will be determined by the task which the module is to perform. (For a more detailed discussion of this system see Stansfield[9].)

7. Conclusion and an Alternate Approach

This chapter has presented several facets of tactile information processing. We began the chapter by presenting some early work in calibration and testing using the LAAS sensor. We then discussed in some detail current work in the design of a robot tactile perception system using the LORD LTS-200 array/vector sensor. This included calibration, the choices of tactile primitives and features, and a discussion of the representation used in the system. From there, we went on to briefly present the next stages of that system: the integration of vision and higher level knowledge toward the goal of object categorization.

Because this chapter is a presentation of tactile information processing in general, and not entirely of our own current research, we would like to end it with a brief discussion of another tactile system developed in the GraspLab: that of Allen[2]. Allen's system, while having the same abstract goals as the ARAMIS system (the combination of vision and touch for object recognition), takes a very different approach to the problem; and we feel that it provides a representative example of alternative tactile information processing techniques.

Allen's work utilized the LAAS sensor mounted on the PUMA 560 robot arm. It's ultimate goal was to match instantiated models against a database of specific, mathematically modelled objects. The representations were hierarchical, feature-based, and were created using a 3-D digitizer and CAD/CAM system. The levels of the hierarchy were: the object level, the surface level, and the patch level (patches were created using bicubic splines.) Features used included surfaces, holes, and cavities.

Initial sensing was via vision. From this vision data, an initial hypothesis about the identity of the object was generated. Tactile sensing was then used to verify the hypothesis. This was done by using the tactile sensor to verify each of the features of the hypothesized object: Holes and cavities were probed; and surfaces were traced to create a finer grid for building the surface patches. Models were matched to the instantiated data at both a local (patch) level and an object level.

The major differences between the ARAMIS system and Allen's are: First, Allen's system contains only one tactile primitive, contact. This primitive is used to obtain geometric points on an object. Secondly, the three features used by Allen's system, surface patches, holes, and cavities, are mathematically defined and are instantiated using both vision and touch; the ARAMIS system defines seperate perceptual systems for both vision and touch, and waits for higher levels of processing to combine the modalities. In addition, the ARAMIS system does not model objects precisely, as does Allen's system, but attempts to model generic instances, thus requiring a more complex control structure. And finally, the ARAMIS system borrows more heavily from psychological theories at all levels of its design. We believe that both of these approaches are valid; and it is interesting to note that they represent, in tactile information processing, the same dichotomy of approach which we often find in vision research.

References

1. Bajcsy, R., D. Brown, J. Wolfeld, and D. Peters, "What Can We Learn From One Finger Experiments?", *A Report on a Joint United States - France N.S.F - C.N.R.S. Workshop on Advanced Automation and Robotics*, 1982.

2. Allen, P., *Object Recognition Using Vision and Touch,* PhD dissertation, University of Pennsylvania, December 1985.

3. Stansfield S., "A Rudimentary, Active, Multimodal, Intelligent System for Object Categorization", Tech. report, University of Pennsylvania, November 1985.

4. Buchner, S., "Installation and Operations Manual: LTS-200 Tactile Sensor", LORD Corporation, 1985

5. Klatzky, R. and S. Lederman, "Hand movements: A Window into Haptic Object Recognition", Paper presented at the 26th Annual meeting of the Psychonomic Society, Boston, Nov. 1985.

6. Lederman, S. and Klatzky, R. , "Knowledge-based Control of Human Hand Movements", Paper presented at the conference on Biomechanics and Neural Control of Movement, Henniker, NH, July 1985.

7. Rock, I., *The Logic of Perception,* MIT Press, Cambridge,MA, 1983.

8. Fodor, J., *The Modularity of Mind,* MIT Press, Cambridge, MA, 1983.

9. Stansfield, S., "Representation and Control within an Intelligent, Active, Multisensor System for Object Recognition", *Proceedings of IEEE Systems, Man, and Cybernetics*, November 1985.

10. Bajcsy, R., *Shape From Touch,* JAI Press Inc, 1985, pp. 209 - 258, ch. 6.

11. Stansfield, S., "Primitives, Features, and Exploratory Procedures: Building a Robot Tactile Perception System", *Proceedings of the 1986 Conference on Robotics and Automation*, IEEE Robotics and Automation, April 1986.

12. Hillis, W., "Active Touch Sensing", Tech. report, MIT AI Lab, April 1981.

13. Klatzky, R., S. Lederman, and V. Metzger, "Identifying Objects by Touch: An Expert System", *Perception & Psychophysics*, Vol. 4, No. 371985, pp. 299-302.

14. Browse, R. and S. Lederman, "Feature-Based Robotic Tactile Perception", *Proceedings of Computer Aided Technology - 85*, 1985, In press

15. Paul, R., *Robot Manipulators: Mathematics, Programming, and Control,* MIT Press, Cambridge, MA, 1981.

16. Overton, K., "The Acquisition, Processing, and Use of Tactile Sensor Data in Robot Control", Tech. report, University of Massachusetts, Amherst, May 1984.

17. Bajcsy, R. and K. Goldberg, "Active Touch and Robotic Perception", *Cognition and Brain Theory*, Vol. 2, No. 21984.

18. Gaston, P. and T. Lozano-Perez, "Tactile Recognition and Localization Using Object Models: the Case of Polyhedra on a Plane", *IEEE PAMI*, Vol. 6, No. 31984, pp. 257-266.

SILICON IMPLEMENTATION OF MULTIPROCESSOR PYRAMID ARCHITECTURE

Franco MALOBERTI
Department of Electronics
University of Pavia
27100 PAVIA - ITALY

1. INTRODUCTION

Pyramid architecture are very promising for parallel image processing [1]. Good interprocessor communication is supported by the pyramid topology and image analysis at different resolution level is allowed. These and other advantages have stimulated the activity of several research groups and now, after theoretical studies, many of then are engaged upon the practical implementation of multiprocessor pyramid architectures. All the realizations must be based on a custom integrated circuit that realizes a defined fractional part of the pyramid.

The implementation of a processor pyramid involves the solution of particular problems that derive by the specific peculiarities of the system. In particular we must remember that the pyramid architecture requires a large number of processors, greater than the one necessary in an array architecture; moreover the intercommunication requirements are large; hence, the physical interconnection inside and outside the chip will determine severe problems.

All these facts have a direct consequence on the definition and on the implementation on silicon of an integrated (LSI or VLSI) circuit that constitutes a subsystem of the multiprocessor pyramid. The technical and practical problems that

NATO ASI Series, Vol. F 25
Pyramidal Systems for Computer Vision
Edited by V. Cantoni and S. Levialdi
© Springer-Verlag Berlin Heidelberg 1986

the designer of the integrated circuit must solve concerns:

- the processor structure
- the pin count limitation
- the dimension of the memory to be integrated together with the processors on the single chip
- the power dissipation
- the testability
- the packaging

In this chapter we will examine all the indicated problems in order to give useful information for the implementation on silicon of a subsystem of a multiprocessor pyramid. Moreover, as a second purpose of this work, it is intended to give to researchers and to designers that are not directly involved on the silicon implementation an awareness on the possibility and on the limits of the to day's integrated technology.

2. CHOICE OF THE TECHNOLOGY

The decision to be made before to start the design of the integrated circuit concerns the choice of the technology. The following preliminary considerations limit the alternative to only the NMOS and the CMOS (p-well or n-well) technology:

- The chip to be designed must be relatively complex; NMOS and CMOS technology have packing density better than bipolar, TTL, ECL and I^2L.
- NMOS and CMOS technology show low power dissipation. This feature is important not only at the chip level but, also, at the system level, since thousands of chips must be arranged on PC boards.
- The speed of CMOS and NMOS technology is not very high but reasonable. With a power per gate of the order of 50 μW and a 3 μ technology, a propagation delay per gate less than 3 nsec results. Hence, a clock frequency of the order to 10 MHz can be easily obtained. This figure is satisfactory for the image pyramid

processing.

- The access to CMOS and NMOS technology is in general easy, in particular for university and research environment.

On the basis of the previous considerations, we assume that the most appropriate technologies for the silicon implementation of multiprocessor pyramid are the NMOS and the CMOS. The selection between the two technologies often depends on practical considerations such as the availability of design tools and libraries of cells or, more in general, the availability of expertise on one of the two selected technologies.

However, if we limits our considerations to technical aspects the CMOS technology would be preferred, because of its lower power dissipation, even if it is less dense than NMOS. The CMOS p-well technology is now largely popular with feature ranging from 2.5 to 5 μm, however, in a next future scaled (1μm or less) CMOS n-well processes will be available; the n-well process is technologically more critical, however it will be preferred since it is faster then the p-well counterpart.

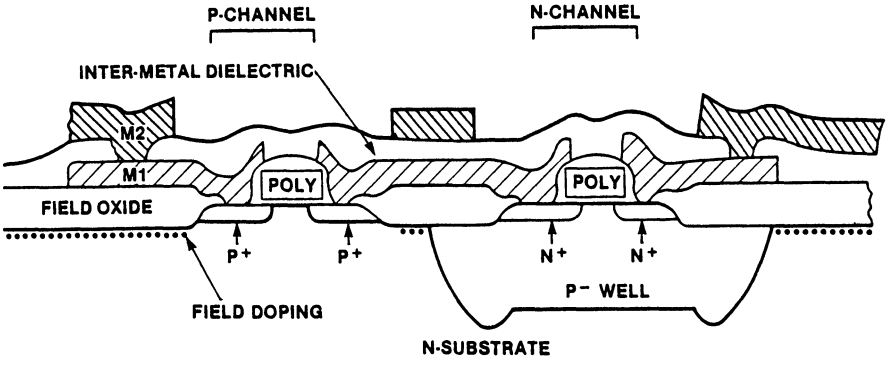


Fig. 1 - Vertical cross section of a double metal CMOS p-well circuit.

An important feature of the technology is given by its number of the interconnection levels. A simple technology has only a single layer of metal; hence, only one level of interconnection is available. The crossings are realized, with short paths, using the (resistive) polysilicon layer. More developed technologies provide two or three

levels of metal for the routing interconnection. Even if the process is more critical, and consequently more expensive, a significant reduction of chip area is obtained. The chip size, compared with a single level metal realization, can be 15% less for the double metal and 30% less for the triple metal realization. The above figures can increase for chips with large interconnection requirements, as is for the pyramidal architectures.

In conclusion a double metal CMOS p-well technology (Fig. 1) seems to be the best today's choice; 1 micron (or less) triple metal CMOS n-well technology the candidate for the future realizations.

3. SYSTEM PARTITIONING

After the choice of the technology and the definition of its features it is necessary to decide how to divide the whole system into identical subsystems to be integrated on a single chip. Of course, the level of partitioning will depends on the minimum channel length allowed by the technology. The considerations of this section will refer to a 2μ MOS technology. It corresponds to the one most widely available for the majority of the researchers in the short term. For more advanced (or conservative) technologies a suitable scaling of the following considerations will be necessary.

A multiprocessor pyramid is a very large system. The number of processor of a pyramid of degree D (the degree is the number of sons of each parent) equals (Dn-1)/(D-1), where n is the number of pixels. Hence, for the parallel processing of a 512 x 512 or a 1024 x 1024 pixel image, using a pyramid of order 4, it results 349,525 and 1,398,101 processors respectively. With such a large number of processors, the main objective of the partitioning of the system is to include the maximum number of processors on the same chip. The limited complexity of the processors (of the order of tens or few hundreds of gates) leads to believe that a large number of processors can be integrated onto the same LSI or VLSI chip.

However, when the number of processors increases, at the same time it increases the requirement of communications inside the chip and, if the pins don't take on a multifunctional meaning (hence reducing the advantages of the parallel or pyramidal processing), it results that the number of processors inside the chip is limited by the maximum number of pins available.

In order to better describe this point, let us evaluate the pin count in a 4D pyramid made with 5, 21, and 85 processors (2, 3, and 4 layers of an elementary pyramid). In the system we will assume that father and sons and neighbors must intercommunicate.

The pin count (using one pin per one function) is detailed in Table 1. It results that a considerable increase of the pin count follows when we move from 5 to 85 processors; the above result imposes important constrains on the packaging and, moreover, has effect on cost, reliability, testability and design flexibility.

Table 1

Pin count of a Pyramid of 5, 21, 85 processors

PIN FUNCTION	5 Proc.	21 Proc.	85 Proc.
Biasing / supply	2	2	4
Instruction set	8	8	8
I / O father	2	2	2
I / O sons	20	80	320
Input neigh bours	12	28	60
Input memory	8	8	8
Testing	1	3	5
	53	131	407

An attractive solution to the problem should be the use of unencapsulated die directly bonded to the rest of the circuit. However, this solution must be excluded, at least in the short time. Die attach to substrate, bonding, testing and providing protection for the finished product are not trivial operations. Die handling must be made by specialists that use expensive equipment and this involves a substantial

capital investment. The only practicable solutions seems to be the use of circuits fully encapsulated in commercial packages. Various kinds of packages are now available with a number of pins that can range up to 144 [2]. Some of them are shown in Fig. 2.

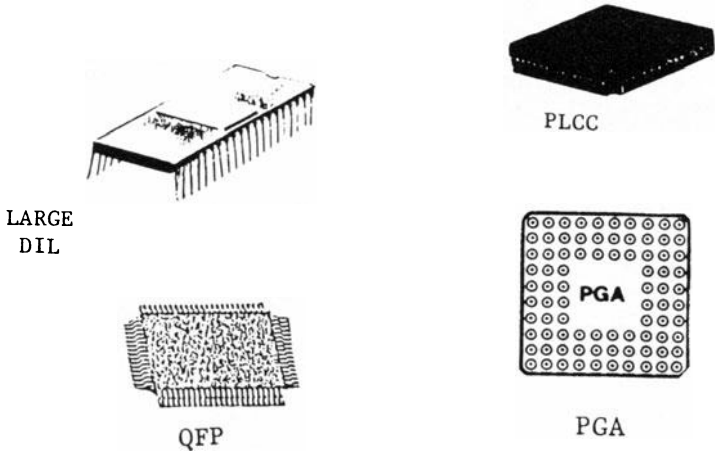


Fig. 2 - IC packages with a large number of pins

The familiar dual-in-line (DIL) package can have up to 68 leads with a spacing between pins of 2.54 mm. The chip carrier family has a pin count that ranges from 20 to 84, with pin spacing of 1.27 mm; although pin counts above 84 can be produced, the problems of handling become severe above 84 pins (the 84 pin chip carrier is approximately 30x30 mm). The PGA (Pin Grid Arrays) package allows up to 144 pins (spacing 2.54 mm). It has an high power dissipation capability and it is now being extended to a low cost plastic version. However the route interconnection on the PC board is difficult and the connections are difficult to inspect. Surface mounting technique also offers a large number of pins. The placement of the IC must be automatic and the soldering technique not conventional. However, the quad flat pack has from 36 to 128 pins with spacing from 1 mm to 0.65 mm. It has potentially a low cost but the number of suppliers is very limited.

An other important point in the evaluation of the pin count concerns the number of pins that must be dedicated to the supply and earth connection. CMOS logic circuits produce short duration pulse currents during the switching action. If the

pulse amplitude is significant, a not negligible drop voltage across the supply buses, that run across the chip, results. A significant effect on the noise immunity of the circuit follows. To prevent this problem multiple supply and earth connections must be considered. Moreover, the pins to destine to the testing of the circuit must be accounted. Their number increases with the complexity of the circuit. For a typical subsystem of a pyramid of processors the number of pin test can range from 1 to 5.

The above considerations on the package, led to conclude that the conventional DIL package is suitable for an LSI multiprocessor pyramid subsystem (eventually with some constrains on the interprocessor communications). For a VLSI circuit, surface mounting or PGA packages must be used. In these cases the miniaturization of the package has the effect of concentrating the total circuit power dissipation into a smaller space. The thermal stress and the thermal cycling increase with consequences on the reliability of the system. Particular care must be paid to this problem, in particular if high speed performances are expected (reader must remember that in CMOS circuits the power dissipation increases with the speed).

4. CHIP DEFINITION

The complexity of a chip, measured in terms of its number of transistors, depends on many factors. The most important are the function that the circuit must implement, the cost of the chip, and the power dissipation. If the function to be implemented needs a large number of integrated circuits, as it is for a multiprocessor pyramid, the designer has available the degree of freedom of deciding the parcelling of the system. Hence, the yield, the reliability and the testing of the circuit can be better optimized. For the specific case under consideration, the complexity of the circuit is related with the number of processors to integrate into a single chip, and this number is usually limited by the pin count of the available packages. Less than one hundred of processors per chip seems to be an upper limit. Hence, with an LSI or a VLSI circuit the number of transistors available can be much larger than the one

required for the processor's design. The resulting redundancy can be used to improve the yield, the testability or the fault tolerance of the chip.

The complexity of the chip is determined by the yield and the cost. These two parameters are interrelated. The yield decreases as the complexity increases; the cost is determined by the area of the chip and by the yield. In Fig. 3 the yield and the normalized cost per chip of a typical CMOS 2 μ technologies a function of the chip size is shown. As results from Fig. 3, the yield decreases of a factor three and the cost increases of about an order of magnitude if the area of the chip increases from 25 mm^2 to 100 mm^2.

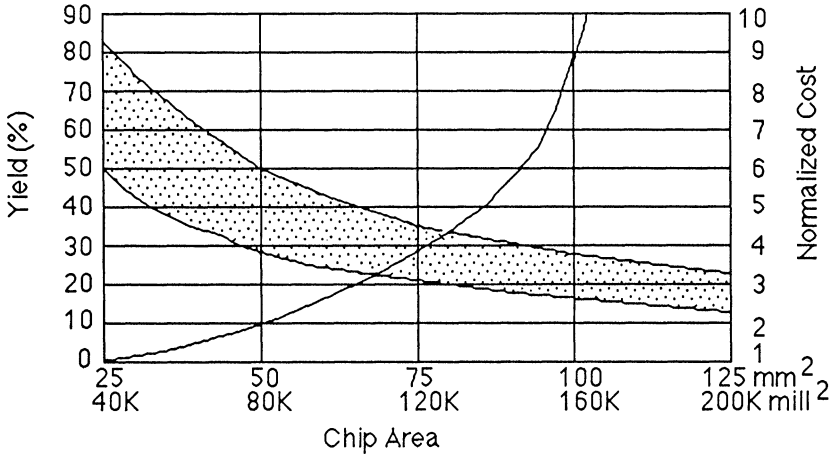


Fig. 3 Yield and normalized cost as a function of the chip area.

A chip can be pad limited. This situation occurs when the frame of the pads determines an active area that is larger than the one required for designing the circuit Usually only very simple circuits are pad limited, however the same situation can occur in circuits that requires a very large number of pads. With a total pitch between the pads of 250 μm, 80 pads can be accommodated into a 25 mm^2 chip and 160 pads into a 100 mm^2 chip. Hence, only integrated sub-pyramid of processors with more than 80 pads can be pad limited.

The total number of transistors that can be designed in a defined active area depends on the randomness of the logic and on the area employed for the

interconnectings; the interconnection area can ranges from 50% to 150% of the area used for the active devices. Usually 0.5 to 3 transistors per $mill^2$ can be designed. The latter figure applies to memories and to very regular structures.

The complexity of a processor to be used in a pyramidal architecture is usually low. In [3] a comparison of the characteristics of several array processor chips (NMOS and CMOS) is shown. The reported data indicate that each processor is made with a number of transistors ranging from 200 to 700. Hence, without any regards to the processor structure, 1000 transistors can be assumed as a sufficient appraisal for the dimension of a bit serial processor. The same number of transistors can be assumed for the implementation of the instruction decoder. Hence, 5, 21 and 85 processors together with the instruction decoder can be accommodated in 4, 15 and 56 mm^2 or less of silicon area respectively. It is evident, at least for the first two considered cases, that the area of the chip can be a bit larger then the indicated value without a significant increase of the cost. Hence, extra circuitry can be cheaply included into the chip. The more pressing request is to integrate memory that, by the way, is compact and take up limited room. 1Kbit of memory per processor can be a reasonable request. However, it is imperative that the designer remember to use part of the available chip area to make easy the testing of the circuit and make it less sensitive to fault. For the latter aspect, the reader is addressed to an other chapter of this book.

.5 DESIGN METHODOLOGY

Design methodology plays an important role in obtaining satisfactory results in a system design. For digital LSI and VLSI circuits various approaches are available. The choice among them is based on economic and project development scheduling considerations. The design methodology to be considered for a pyramid architecture are the gate arrays, the standard cells, the PLA and the parametrized cells (silicon compiler) approaches.

A gate array, or uncommitted logic array (ULA), is an integrated circuit that incorporates, arranged in rows and columns, thousands of transistors or simple gates (NOR or NAND). The integrated circuit wafers are preprocessed in advance till to the masks defining the interconnection levels. The customer must design only these masks that are applied to the gate array wafer. Hence, a quick and inexpensive development of a chip results. Several companies have developed gate arrays families ranging from few hundreds to many thousands of equivalent gates. The design of the masks describing the inteconnection level is often performed with user-oriented CAD tools that support the various phases of the design, in order to ensure fast turn-around cycle times. These CAD tools are: logic simulators, used to confirm that the design performs the desired function; timing simulator, used to verify the operation at the desired speed; placer and router, used for the automatic determination of the best location of the gates and the best path of interconnections. Since the gate arrays have a defined size and, moreover, the gates can not be completely routed, the gates are not completely all used. A typical efficiency is 75% with maximum values of 95%. Fig. 4 shows a typical (500 gates) gate arrays. Routing channels and rows of cells are evidenced together with a chip photograph [4]. Gate arrays are efficient for the design of random logic but, for the implementation of memories the technique shows serious limitation.

The standard cell approach is based on the use of a library of logic elements from which customized chips can be easily designed. The cell library contains basic cells (NOR and NAND), more complex elements (exclusive-OR, flip-flop, binary full adder, etc.) and memory cells. The design process consists in the selection from the library, the placement on rows and the interconnection of the necessary cells. The advantages with respect to the gate arrays result from the availability of complex cells and from the possibility of adjusting the width of the interconnecting channels. Moreover the chip size (if the circuit is not pad limited) is determined by the complexity of the circuit. Fig. 5 shows the layout of two cells (CMOS-3μ by SGS). It can be observed that the cells have the same height, moreover on the top and on the bottom of the cell the supply and the earth lines run. In this way a row of cells can be composed simply by placing cells side by side. The input and the output connections are in the direction orthogonal to the supply lines.

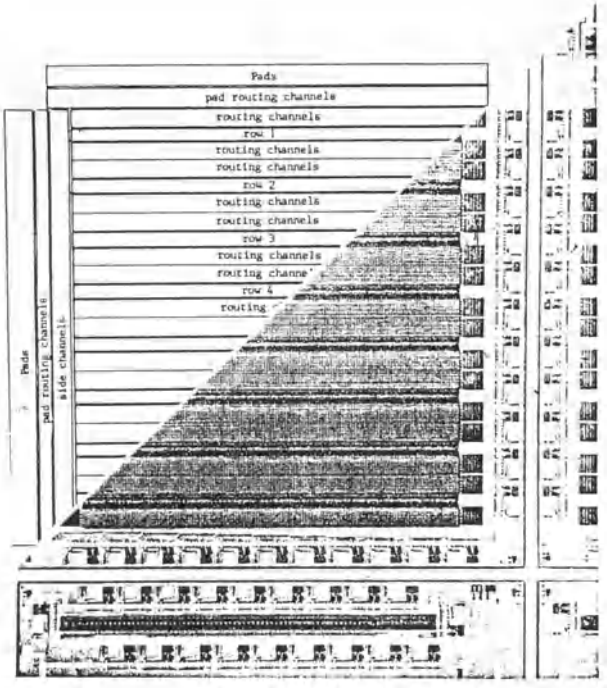


Fig. 4 - Layout of a gate arrays (500 gates)

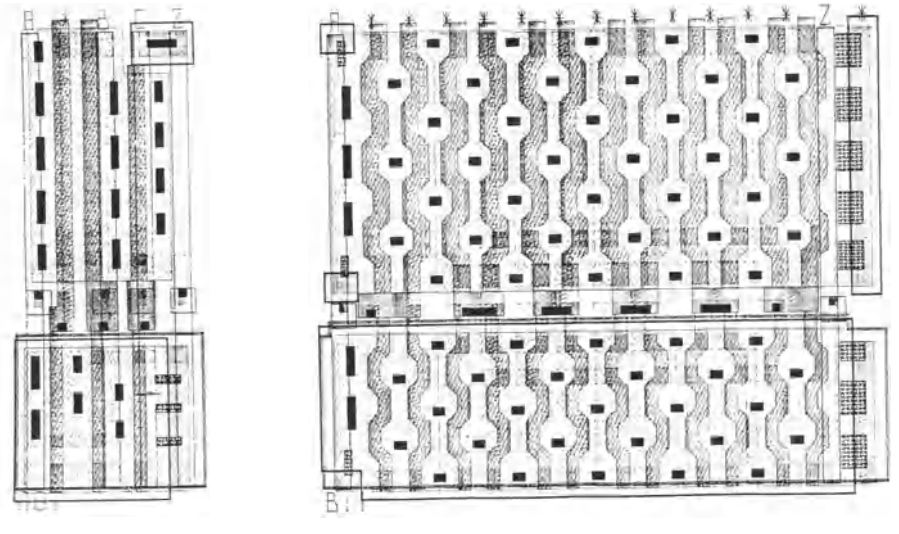


Fig. 5 - Layout of two standard cells (CMOS 3µ) [Courtesy of SGS]

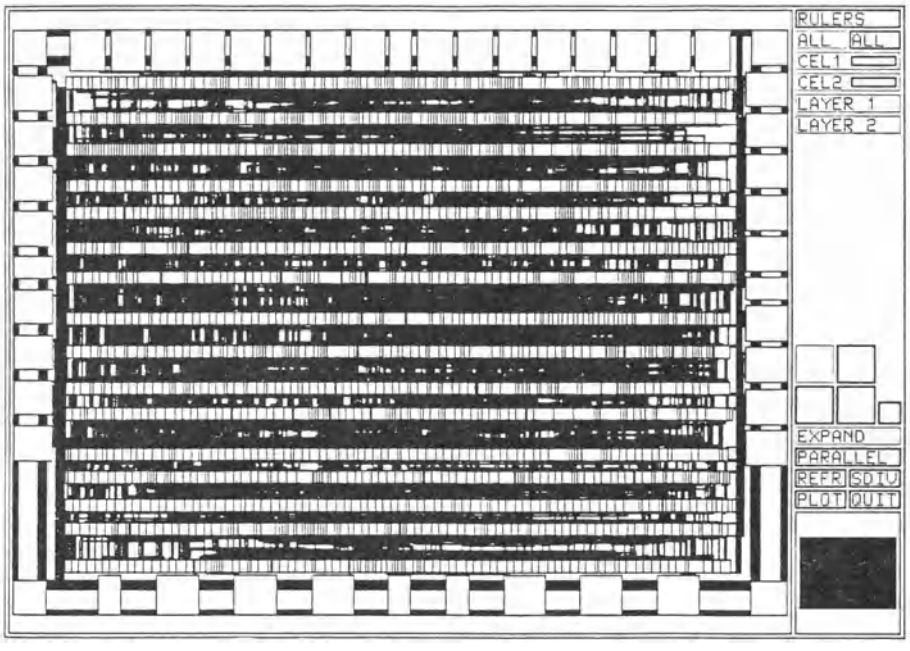


Fig. 6 - Layout of a 3μ CMOS standard cell. (SGS)

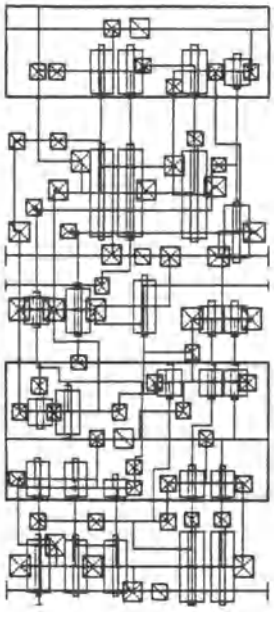


Fig. 7 - Stick diagram of a CMOS cell (SYMBAD - SGS)

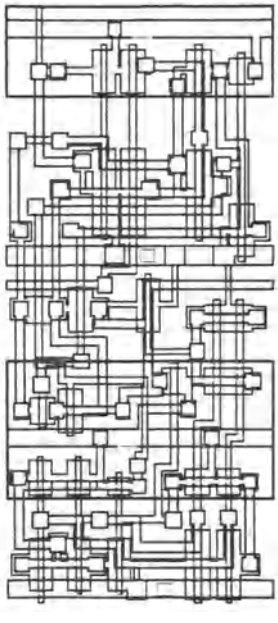


Fig. 8 - Compacted CMOS layout, automatically generated by SYMBAD

The chip area of a standard cell can be as low as 40% (or more) of an equivalent gate arrays. As an example, consider the 1701 cells 3μ CMOS standard cell chip shown in Fig. 6 [5]. It contains 24 different type of cells; the area occupied by the cells is 10.33 mm^2, the total active area is 26.11 mm^2 (interconnections overhead is 156%), and the total area of the 45 pad chip is 35.21 mm^2. The circuit is a redesign of a 3μ CMOS gate array. The used gates ware 3738 out of 4200 (efficiency 89%), the total chip area was 78 mm^2. The design of the library standard cells can be made by hands or by the use of CAD tools. An example of design tool is given by the program SYMBAD, by SGS [6]; the program uses a symbolic description (stick diagram) (Fig. 7) of the cell and automatically designs the CMOS layout compacted to the minimum feature allowed by the technology (Fig. 8).

Programmable logic arrays (PLA) allows to realize arbitrary combinational logic functions through the transformation in the so-called sum of product form [7]. The basic advantage of the PLA results from its highly regular structure that easily allows the automatic generation of the layout. Various programs ware developed for the layout of PLA and for the minimization of the chip area by the so-called folding operation.

The last design approach is the design and the assembling, assisted by CAD tools, of parametrized cells. This design methodology is also known as silicon compilation [8]. In a narrow sense, the method is an extension of the standard cell approach. The position of some I/O ports on the boundary of cells are automatically modified in order to avoid interconnections in the connecting channel and, hence, to reduce the interconnection overhead. To this option other CAD tools for the design of microarchitectural components where added in order to get an integrated package of IC design. However, blocks that are designed by different approaches have different height and some of them (for example PLA) do not have a rectangular shape; this makes complex the automatic interconnection of blocks. Recently CAD tools that allows the automatic placement and routing on nonrectangular blocs ware developed. An example is given by the program MINERVA [9]; a typical output of MINERVA is shown in Fig. 9.

The automatic sequential use of the described design tools performed using the same data base can allow the translation of a high-level description into a layout.

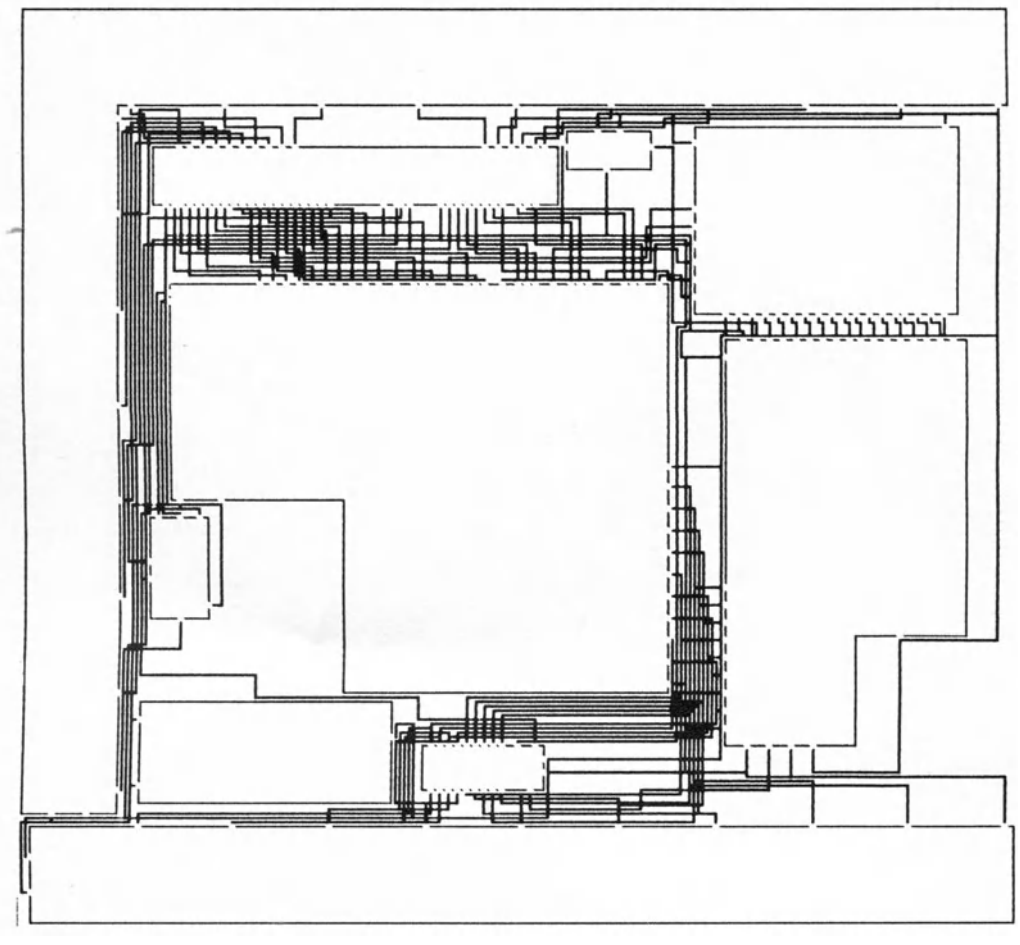


Fig. 9 - Placement and ruoting of non rectangular blocks by MINERVA (SGS)

This process is, in a broad sense, the silicon compilation.

.6 TESTING

The growth of the device complexity and the increase of the pin count have an important impact on the problem of the testing. A complex chip is difficult to test

because the internal gates become more 'buried' and thus it is more difficult to control their input and to observe their output. The right operation of the circuit is verified with testes that load in parallel into the input pins suitable test patterns and verify the outputs. Commercial testers have I/O pin counts limited to 128 and 10 MHz speed. Even if the tester technology is improving (256 pins and 40 MHz performances are coming), the designers must be aware that their designs must be compatible with the production tester.

An important point concerns the test pattern generation. The correct behavior of the circuit is verified by developing several sequences of input patterns and the responses are then compared with those expected. To prove the functionality of any logic element within the design it is necessary that all its inputs be exercised by sequences of test pattern applied to the input pins (controllability) and that all its output signal reaches an output pin (observability). As the complexity of the circuit increases it is more and more difficult to observe and to control all the internal logic elements of the design with a reasonable number of test patterns. Hence, it may result convenient to add extra hardware that improves the controllability and the observability of the circuit (design for testability).

Various approaches of design for testability exist. One is the "ad hoc" testability approach, which consists of circuit partitioning and added test points. Another approach is the "scan path", which consists in the reconnection of sequential elements to form a shift register that allows to easily control and to observe internal elements. The circuit added to improve the testability requires a very low overhead in terms of chip area (few %) and can significantly reduce the cost of the testing of the chip. For complex circuits three or more pins for the testing can be required.

7. CONCLUSION

This chapter has attempted to illustrate the problems related to the silicon implementation of multiprocessor pyramid architecture. It must be stressed that the

developed considerations mainly concerns the today's problems. This because it is intended to give a contribution to the present realizations rather then furnish indications on future developments.

REFERENCES

[1] V. Cantoni, M. Ferretti, S. Levialdi, F. Maloberti (1985) Chapter Eight of: S. Levialdi "Integrated Technology for Parallel Image Processing" Academic Press

[2] B. S. Cowle (1985) "The cost, quality and system design aspect of current and future integrated circuit options"; Proc. of the 2nd Silicon Design Conference, Wembley, pp. 59-66, 1984

[3] T. J. Fountain (1983) "A survey of Bit-Serial Array Processor Circuits"; Chapter One of: M. J. B. Duff "Computing Structures for Image Processing" Academic Press

[4] R. Percival (1984) "LASARRAY - a quick turn-around maskless gate arrays"; Proc. of the 4th Int. Conf. on Custom and Semi-custom ICs, London;1984

[5] SGS Microelettronica - Private communications

[6] A. Montrezza, L. Sturlese, D. Devecchi, M. Morganti, G. Torelli:"SYMBAD: a Tool for New Layout Methodology" Proc. CICC, 1985, pp.64-67

[7] D. Hodges, Jackson: "Analysis and Design of Digital Integrated Circuits"; Addison Wesley, 1984

[8] D. D. Gajski: "Silicon Compilation", VLSI System Design, nov. 85, p. 48

[9] MINERVA, User Manual, SGS - 1985

FAULT-TOLERANCE TECHNIQUES IN ARRAYS FOR IMAGE PROCESSING

Roberto NEGRINI - Renato STEFANELLI
Department of Electronics - Politecnico di Milano
P.zza L. da Vinci 32, I-20133 Milano - Italy

1. SUMMARY

This paper treats fault- tolerance in arrays for image processing. First, the basic methodologies of fault- tolerance that can be used to reconfigure arrays, allowing them to survive faults and defects, are examined. Main figures of merit of reconfiguration algorithms for fault-tolerance are then treated. Then, known reconfiguration algorithms are recalled, and discussed. These algorithms have been defined for the self-reconfiguration after fault of rectangular arrays (meshes) where cells are connected to the 4 nearest neighbors by means of unidirectional communication paths. These algorithms are then extended to the case of bidirectional meshes typical of image processing; extension to pyramids is also considered.

2. INTRODUCTION

Many different kinds of arrays for image processing have been defined in the past. In spite of such differences, some classes having common characteristics can be identified.

Here, we will focus on the many structures that are based upon large two- dimensions arrays (e. g. more than 128*128) or large pyramidal arrays of rather simple processing elements (also called "cells"), where each element is linked only to a small group of "nearest neighbors", e. g. the four adjacent elements in two- dimensions arrays. This communications locality allows for very high circuit integration (very low area requirements for interconnections).

Usually, these arrays are SIMD machines, where all the processing elements execute contemporarily the same instruction, so that there is only one program under execution at each time, and each processing element operates, synchronously, upon its proper subset of the data that describe the images under processing. Data of a processing element are usually stored in a small local memory (e. g. 256 bit). Instructions of the program are sent in parallel to all the cells of the array, by a controller that can be a dedicated microprogrammed unit, or (and with more user-friendliness) a quite standard computer that uses the array as a dedicated, powerful arithmetic and logic unit.

Due to limits of present technology, two main implementation classes can be identified:

1. structures based upon VLSI custom integrated circuits containing few processing elements (e. g. 4*4 or 8*8 in the case of two- dimensions

---

This work was partially supported by grants from NATO and from Centro Volta, Villa Olmo, Como - Italy

NATO ASI Series, Vol. F 25
Pyramidal Systems for Computer Vision
Edited by V. Cantoni and S. Levialdi
© Springer-Verlag Berlin Heidelberg 1986

arrays);

2. structures adopting Wafer Scale Integration (WSI).

The large amount of processing elements is necessary in order to obtain the required computing power, very high when real- time applications are foreseen. Adoption of a very large number of elements causes a common problem, namely the high probability that some of the elements can be faulty or defective.

Thus, the goal arises of adopting methods for reconfiguring arrays containing faults, in order to build up either the largest possible working array, or an array of given fixed dimensions - starting from a large set of processing elements.

The solution to this problem depends on the characteristics of the faults that in practice can hit the arrays. A first distinction can be made between defects introduced at production- time, and faults arising on the field, for noise, electrical stress and so on.

Defects limit the production yield: they are important for very large VLSI integrated circuits, and extremely important for WSI implementation. Standard fabrication processes introduce 1-10 localized defects per square centimeter of silicon area. In case of WSI, also large defective zones can be produced. Analyses upon process defects can be found e. g. in [1] and in later papers by Stapper on the same journal; in [2] a large survey of fault- tolerance techniques for enhancing circuit yield is given.

WSI arrays contain much more defective cells than VLSI arrays (for end-of- production screening can reduce nearly to zero the number of defective cells), and defective cells in WSI can also appear grouped in clusters. These differences between VLSI and WSI can influence the choice of reconfiguration algorithms. For example, reconfiguration algorithms for VLSI arrays can be based upon few redundancies (e. g. spare cells), whereas algorithms for WSI must tolerate cases where e.g. 80% or more of cells are defective.

In this paper, we will deal both with algorithms specialized for reconfiguration of VLSI and of WSI arrays. In the next section, the basic problems of reconfiguration for fault- tolerance will be treated.

Then, the basic figures of merit that can be defined for reconfiguration algorithms will be discussed. Influences of array structure upon suitability and ease of adoption of fault- tolerance will be examined.

The following sections consider many basic reconfiguration strategies that can be the base for different classes of reconfiguration algorithms. Many reconfiguration algorithms, mainly useful for two- dimensions arrays, are then shown.

3. PROBLEMS AND APPROACHES TO FAULT- TOLERANCE

Many problems must be solved when aiming at introduction of fault-tolerance in arrays. These problems have very different characteristics, depending on:

1. when reconfiguration is done, and
2. which is the controller of the reconfiguration.

Static reconfiguration can be actuated at end of production: in this case, only production defects can be overcome. It can also be actuated dynamically during the array operating life, on the field, and at different times (e. g. just each time the array is powered, or from time to time at fixed periods, or upon operator demand, or also whenever a new fault is

automatically discovered in the array). In this second and more demanding case, also operational faults can be eliminated.

As for the second point, usually two possibilities arise.

- The reconfiguration can be host- driven, i. e. actuated by the external controller of the array, or
- the array can be self- reconfiguring, i. e. capable of identifying faulty cells and of rearranging links among working cells (in order to reconstruct a perfectly working array) with no external intervention.

In every case, it is necessary that all the faulty cells of the array are perfectly identified.This introduces the problem of testing the array; in case of dynamic reconfiguration, tests can be executed by the controller of the array, or autonomously by the array itself (self- testing). Identification of faulty cells not strictly necessary if the array is used to implement only "robust" application algorithms, inherently redundant, that produce also redundant results that the user can process further, eliminating the erroneous parts. This kind of application algorithms is not common in the field of image processing that adopts the arrays here considered: examples can be found in some special cases of matrix arithmetics.

In case of dynamic reconfiguration, an important characteristic of testing methods is the latency time of a fault, i. e. the time that passes between the moment that the fault starts to corrupt data, and the moment that a check identifies its presence (even if, possibly, not yet identifies its place in the array, i. e. the fault is not yet fully diagnosed).

This clearly introduces also the problem of effectiveness of testing (i. e. the probability that a check does not find a fault) and of diagnostic actions if separate from testing (i. e. the probability that a fault is attributed to a wrong cell).

After diagnosis of faulty cells, it is always necessary to exclude them from the array. A first kind of exclusion is physical isolation, necessary to prevent that electrical or thermal stresses propagate to nearby cells. This isolation can be only very partial, if done by logic circuits, in an automatic way.

A second, usually easier kind of exclusion is logical isolation, preventing that erroneous data generated by faulty cells arrive to working cells, and preventing also that faulty cells disturb normal communications and data communication among working cells.

It seems that, being the cells of arrays here treated fairly simple and not separately programmed, self- testing and isolation methods similar to those identified in [3] cannot be applied. These methods are based upon automatic constitution of "clusters" of cells testing each other, and voting upon test results, in order to establish a "consensus" among working cells that correctly state which cell is faulty.

Depending on the various kinds of parameters and choices considered, reconfiguration algorithms introduce overheads: additional circuits are necessary, and throughput can be reduced, mainly for the necessity of having testing periods.

Additional circuits are necessary:

- to test the cell. These circuits implement self checks, if required, or temporarily isolate each cell, making it possible to apply external testing and diagnostic procedures. Self- testing can be made in many different ways, based upon duplication of circuits and checks of consistency of results, or upon adoption of robust logic circuits,

signals and data (quadding, double rail logic, error detecting coding of data, etc., see [4]).

- to disconnect the cells, and to introduce new connections. In case of in field reconfiguration, these operations can require the presence of many alternative paths linking each cell to a small set of nearby cells, in order to dynamically choose the few neighbors that have to be connected in order to build the array. In case of end of production configuration, this operation can be done by static and irreversible means, as e. g. disconnecting by laser cutting or fuse blowing, and connecting by laser welding or through inverse fuses.
- to drive the reconfiguration actions, in case of dynamic self-reconfiguration.
- (if necessary) to properly roll back to previous correct values of data contained in each cell, or to reconstruct new acceptable values. For example, this can require adoption of extra copies of memory modules in each cell, in order to maintain a data backup, or it can require extra links among cells, in order to allow that a cell accesses data contained in the memory of a neighboring cell. This problem also arises only in case of in field reconfiguration, particularly when self-reconfiguration is foreseen, and the operation must be completely hidden to the systems that are connected to the array.

Spare cells also can be introduced in the array, in order to substitute the faulty ones. This is particularly true when an array of given, fixed dimensions must be built. Actually, two "kinds" of spare cells can be adopted.

In the first case, real spares can be added to the array. If no faults are present, and if the goal is to build an array of given dimensions, then these spare cells are not used at all. This form of redundancy is usually called "space- redundancy" (or circuit- redundancy).

In the second case, "time" redundancy is adopted. All the cells, synchronously, proceed through at least two alternating phases. In the first phase, all the cells perform their nominal functions. In the second one, the cells are free; thus, in this phase the working cells can perform also the functions of nearby faulty cells.

The amount of allowed overhead circuits depends on the dimensions of the basic cell. The larger is the cell, the larger can be the added circuits. This is due to the fact that, as suggested in [5], added circuits and spares can also be affected by faults; incrementing too much their relative area reduces the true reliability. Too many spare cells also can have a global negative influence: they can require many spare links occupying large areas, where more operational circuits could have been put. Furthermore, defects can hit these spare links also.

4. FIGURES OF MERIT OF RECONFIGURATION ALGORITHMS

"Costs" of given reconfiguration algorithms can be stated in terms of

- additional circuit area
- additional processing time due to additional delays of data propagation through the interconnection network, or to time spent for checks, as previously discussed.

Another negative figure of merit is the time wasted for reconfiguration actions after that a new fault is diagnosed. Actually, this figure of merit is not very important in practice. It is not important for end of production configuration, and it has a light importance for in field

reconfiguration (mainly for automatic configuration at power up). It might have importance in case of repetitive transient faults, as it happens in presence of noise. But in this case other more negative facts have primary importance, like necessity of restarting computation. In these cases, usual assumptions foresee the "staticization" of transient faults (transient faults are treated as permanent).

As we will see, practical reconfiguration limits (introduced when only few spare links can be used for reconfiguration) make it possible that not all the working cells can be linked together to form the working array. In other cases, some working cells are wasted and used as connecting elements (relays) only, in order to allow communication between distant cells, and do not operate upon data passing through them.

In other words, given a fixed structure of interconnection spare paths (and thus a fixed amount of extra circuit area), not all the reconfiguration algorithms allow for 100% "harvesting" of working cells. Worse, not always a reconfiguration algorithm is capable to reconstruct an array that maximizes harvesting. Other points of interest are mean and maximum single connection length: these two parameters clearly influence propagation delays of data communications among cells, and thus have importance in determining performances of the reconfigured array. They are less important in case of WSI arrays, where signals can cross the wafer along a metal path with delays that in some cases can be ignored in comparison to circuit delays, and to transit delays through disconnect/ interconnect circuits or through relay cells.

Thus, it can be possible to introduce the concept of "optimality" of a reconfiguration algorithm, where optimality means maximization of one or more of the discussed figures of merit.

Not always all these figures of merit can be analytically computed. Often, it is necessary to adopt statistical evaluations. Software simulators can be built for each reconfiguration algorithm, and many randomly generated reconfiguration problems solved, determining by inspection the required figures of merit.

It must also be noted that the importance of the various figures of merit heavily depends on real- life defects and faults that can hit a given array. Thus, comparisons among algorithms are possible only within clearly defined application frames, identifying such application parameters as cost of down time, costs of erroneous processing, value of correctly obtained data, cost of maintenance, and so on. This problem is also complicated by the fact that all these data can be uncertain, not defined, or largely variable.

5. ARRAY STRUCTURES AND RECONFIGURATION STRATEGIES

The array structure largely influences possibility of introducing simple and effective reconfiguration strategies.

First, it is necessary to distinguish between synchronous or asynchronous arrays. Synchronous arrays introduce more constraints into reconfiguration; for example, clock frequency can depend upon length of the longest link between cells after reconfiguration and upon delays introduced by relay cells along links.

The presence of relay cells can constitute a problem in synchronous arrays, since these cells also at least influence maximum propagation delays along paths. If relay cells can not act as combinatorial circuits, but latch data flowing through them, this causes a far worse problem: they introduce a clock step delay and (in order not to loose the perfect synchronization of the array) the same step delays have to be introduced in

suitable positions along other communication paths, even far ones, and this fact practically inhibits adoption of self- reconfiguration algorithms. In the sequel, we will always suppose that all the relay cells are bypassed by combinatorial circuits characterized by very low delays.

These difficulties are not present in case of asynchronous (or self-synchronizing) arrays, like the "wavefront- computation" arrays. In these arrays, differences of propagation delays along the various links are acceptable; they influence throughput but do not introduce errors in the computation. In these arrays, a cell starts to operate only when its previous computation is completed, and all the new data necessary for the new computation are present at its inputs. This can be implemented through handshaking signals at each link.

A second important difference exists between arrays of sequential cells or of combinatorial cells.

This difference becomes important after reconfiguration, if results of previous processing, stored in registers or memory at each cell, can not be lost.

Actually, this problem can be present in a different form also in the case of combinatorial cells, but it is then confined inside the host system controlling the array, where processed data are kept, and does not affect reconfiguration algorithms and overhead circuits of the array.

In case of sequential cells, different requirements arise if memory or registers of a cell can (or can not) be accessed by nearby cells, and if it is reasonable to assume that a cell fails independently from its memory and registers. In this case, it is then possible to adopt methods similar to those introduced for microprogrammed control units [6]. For example, it is possible to introduce backup memory and registers, where from time to time the status of the array is saved. After reconfiguration, the last saved status can be restored and processing rolled back to that point. The usual problems of roll- back methods (correct identification of the moment when the actual status must be saved, effects of faults hitting backup memory, etc.) are present here also.

It can be said that the case of sequential cells is very complex, if roll- back is necessary. Otherwise, it is only slightly more complex than the case of combinatorial cells.

A third important point refers to the geometric structure of the array.

Linear (one- dimension) arrays can be easily reconfigured by means of methods that easily guarantee optimality in harvesting and length of restructured links. Also synchronous arrays can be easily reconfigured; clock skew problems also are easy to solve. Many reconfiguration algorithms exist capable of reconfiguring these arrays.

The case of two- dimensional arrays, like trees or meshes, is complicated by the necessity of maintaining two- dimensional structural constraints. This fact, in many cases, keeps algorithms far from optimality, and not all the working cells can be used.

Of course, reconfiguration algorithms valid for two- dimensions arrays are also capable of reconfiguring one- dimension arrays.

Algorithms for reconfiguration of meshes will be treated in detail in the next sections, for the case of cells having 2 inputs and 2 outputs (rectangular meshes with unidirectional data flow). Then, the case of meshes of cells with 4 inputs coming from the 4 nearest neighbors and 1 output (broadcast to all the same 4 neighbors) will be considered.

The problem becomes more complex for three- dimensions arrays, like pyramids. It is possible to define methods that map three- dimensions arrays onto one- or two- dimensions arrays, and then adopt reconfiguration algorithms for these simpler arrays. Usually, these methods are complex and more suitable for end of production configuration.

It is also possible to try to extend methods valid for two- dimensions arrays (e. g. reconfiguring separately each plane of a pyramid, and then connecting in the right way cells on adjacent planes). In the sequel, we will follow this second way, trying to modify the methods valid for meshes that we will consider in the next sections.

All the reconfiguration algorithms can be divided in few classes, adopting different reconfiguration strategies ranging from very simple ones (suitable for self- reconfiguration), to complex ones, more suited to end of production or host- driven implementation.

In general, these strategies require that the working condition of each cell is asserted. Usually, each cell is assumed to be either "Faulty" or "Perfectly Working".

The reconfiguration strategy that is the simplest to implement automatically, and cost- effective in case of two- dimensions meshes, is based upon elimination of complete rows or columns.

When a cell fails, its column (or row) is completely bypassed, and a spare column (or row) is activated on the border of the mesh. Of course, this strategy is good only for very low probability of faults (thus, for VLSI and not for WSI arrays, and with a good production screening), and for in field reconfiguration (even self- reconfiguration), for the extreme simplicity of required reconfiguration circuits. In fact, this strategy requires only, at each cell, a local spare path necessary to bypass the cell. This strategy is adopted in [7].

More complex strategies eliminate "cut sets" of cells more complex than one row or one column. For example, in one of the methods of [8], valid for two- dimensions meshes with unidirectional data flow, two "bypass" registers are added to each cell (one for rows and the other for columns). If a cell is faulty, its bypass registers are activated, and the cell is isolated; data passed to this cell are simply stored in the bypass registers, and then forwarded at the outputs. In order to reconfigure the array, a cut set crossing it is defined, composed by working or (better) faulty cells: all the cells in the cut set are also substituted by their bypass registers.

Data links that cross the cut set are rearranged in order to maintain the mesh structure, by means of spare paths. For example, the simplest approach consists of changing from vertical to horizontal all the vertical paths that enter the cut set.

This procedure allows for reconfiguration of synchronous arrays, like the two- dimensions "systolic" ones. Identification of an "optimal" set of cut sets (e.g. that maximizing harvesting) is not an easy task, and requires implementation either by host controller or off line, at end of production.

More complex and better suited to off- line reconfiguration, other strategies partition the array, and reconfigure independently each part. At the end, the various parts are connected to form the complete array.

For example, a two- dimensions mesh can be partitioned in two parts containing the same number of working cells. Recursively, the method can be iterated upon these parts, till 2^k very small and easily reconfigurable sub- parts are obtained. This allows configuration of 2^k sub- meshes, all

of the same dimensions. Then, partitioning is back- tracked, joining sub-parts that were split, ordinately linking together corresponding cells on the borders of the splitting line.

Partitioning can be adopted in a fixed way also. In this case, the physical array is divided into fixed (equal) blocks; among the blocks, bundles of wires can be used to interconnect the blocks. Every block can be recursively partitioned again, otherwise inside it a different reconfiguration strategy can be adopted (see [9] as example).

Another strategy for off- line reconfiguration of two- dimensions meshes is based upon mapping binary trees over the mesh, as described in [10]. Each node of the tree defines a sub- mesh, whereas the root defines the required complete mesh, and leaves correspond to working cells in number sufficient to construct the mesh. The tree can be projected over the set of working cells, thus identifying both working used cells and connections among them.

Two other basic strategies will be discussed in detail in the two next sections. Both are particularly suited to self- reconfiguration of two-dimensions meshes, characterized by unidirectional data flow. In both cases, only a small number of spare interconnection paths are necessary at each cell. The next two sections will depict algorithms that adopt these two strategies, discussing their properties and requirements.

It can be noted that all the required spare paths, for both these two last strategies but also for the previous ones, can be practically implemented in different ways.

A first implementation can be based on the presence of all the spare paths needed, in their completeness. The correct path can be chosen by means of input multiplexers at each input of the cells. Dynamic reconfiguration of paths can be obtained, by sending selection signals to the multiplexers.

A second implementation can be based upon buses that cross all the array, in horizontal or vertical channels among the cells. In this case, links between cells can be routed through the buses, connecting horizontal and vertical buses where they cross, and cutting buses where necessary (a form of channel routing similar to that used in many gate arrays). This schema can be used to either dynamically or statically reconfigure paths, depending on the adoption of reversible or irreversible means for connecting and cutting buses.

A third implementation also can adopt horizontal and vertical buses, but at each crossing it is possible to insert, if necessary, a switch capable of connecting the four bus ends in all the possible ways. In this case, paths are built by connecting segments of buses. Constraints in setting connections at the switches (if not all the configurations are possible, or if few segments are allowed) can make the reconfiguration more difficult than in the previous case.

The second and third implementation methods require less spare paths than the first one, but are usually more difficult to control.

6. SELF-RECONFIGURATION OF VLSI MESHES

In this section, we discuss principles of algorithms tailored for reconfiguration of VLSI two- dimensions rectangular arrays, where few spare cells are added along the borders (e.g. one spare column at the right edge and/or one spare row at the bottom). Only "space- redundancy" will be here considered.

The basic approach is that, given a fault distribution, faulty cells are not directly substituted one-to-one by spare cells: for each faulty cell, an ideal line is identified, that starts from it and ends to a working cell, and passes through working cells only. Reconfiguration is obtained by "shifting" the cells along this line, so that the nominal functions of each cell of the line are executed by the following cell in the line.

If the line ends at a spare, this spare is activated and reconfiguration is completed. But the line might also end at a working cell which is not a spare: in this case this cell is practically a "pseudo" faulty cell (or, in other words, it is "stolen" in order to act as a spare), and another reconfiguration line must start at it. In any case, the complete line, composed by various segments, must always start at a faulty cell and end at a spare.

The various reconfiguration algorithms differ in the way that the reconfiguration lines are identified and built. A fatal failure corresponds to the impossibility of building reconfiguration lines for all the faulty or stolen cells.

Families of algorithms generated by this approach have been presented in the past (see e.g. [11], [12], [13]).

In the simplest case, only a spare column is added at the right border. Then, a very effective policy can be identified: for each faulty cell, its reconfiguration line follows the row towards right till it ends at the spare cell of the row, at the rightmost position. Reconfiguration control networks are very small, and few very short spare paths are required. In practice, each cell must be linked either to the upper and lower nearest neighbors on its column, or to the two corresponding neighbors on the column at the left. These two spare links are activated in all the cells at the right of a fault.

Fatal failure occurs whenever two or more cells in a row are faulty: this condition can be easily detected and signaled, but it limits survival probability.

More complex solutions, tolerating two or more faulty cells on the same row, and then characterized by higher performances, can be found in [12]. They require spare paths for horizontal links also, and more complex control networks.

A formal definition of reconfiguration algorithms will be now discussed: it will also be used to describe one of the algorithms of [12].

6.1. A formal approach to VLSI reconfiguration algorithms

If, for each cell, we define:

- "physical" indices (i,j), identifying position in the physical array
- "logical" indices (i',j'), identifying position in the logical reconfigured array consisting of working cells only,

then the reconfiguration may be described by an algorithm mapping logical onto physical indices.

A N*N array with one spare row and one spare column is described by a (N+1)*(N+1) physical index matrix P(i,j) where PH(i,j)=i,j.

A second (N+1)*(N+1) matrix is the logical index matrix LG; entry LG(i,j) is the pair of logical indices $LG_x(i,j)$, $LG_y(i,j)$ associated with cell (i,j). If (i,j) is not active (it may be either faulty or an unused spare), both logical indices are set to 0.

A "fault matrix" F(i,j) describes the fault distribution, being F(i,j)=1 or 0 if cell (i,j) is faulty or working.

A reconfiguration algorithm can be described as an ordered sequence of applications of suitable operators that, given a fault matrix F and an initial logical matrix LG (e.g. equal to PH), allow to obtain the final logical matrix LG. Each operator acts on a "deformation matrix" derived from F and on the "present value" of the logical index matrix LG, identified by LG' (for the first operator of the sequence, LG' is the initial logical matrix) and it produces the "next value" of the logical matrix LG, identified by LG".

Definition of two operators RR and DD is now given.

1. A right- reconfiguration operator RR acts upon a right- deformation matrix R and upon LG' so that
   - LG"(i,j)=LG'(i,j-1) if R(i,j-1)=1
   - LG"(i,j)=LG'(i,j) if R(i,j-1)=0 and R(i,j)=0
   - $LG''_x(i,j)=0$, $LG''_y(i,j)=0$ if R(i,j-1)=0 and R(i,j)=1

     (being LG_x the logical row index and LG_y the logical column index)
2. In a similar way, a reconfiguration operator DD of the vertical, bottom direction may be defined, by interchanging i and j.

These operators are not commutative, i.e. the final logical index matrix LG depends on the order of application of the operators.

A specific algorithm is then created by defining: a) the operators to be applied and their ordering; b) the deformation matrices.

Now, an algorithm will be described, that adopts one spare column and one spare row. In this algorithm, the rightmost fault in a row is reconfigured at right using the spare cell of the row, as in the simple case previously discussed; the other faults of the row simply steal neighbors of the lower row. These stolen cells act as faults in their row. Fatal failure arises, for example, whenever a cell cannot steal its lower neighbor (because it is faulty) and also cannot reconfigure towards right - for example because there is a faulty (or stolen) cell in the part of the row at its right.

This algorithms will be called "simple fault-stealing, fixed choice".

Assuming one column and one row of spares, entries in the deformation matrices are defined as follows:

- matrix F is scanned from row 1 to row N; assume that in row i there are k faulty cells such that $F(i,r)\neq 0$ ($r=j_1 \ldots j_k$ and $j_1<\ldots<j_k$), then:
  - $R(i,j) = 1 \quad j_k \leq j < N+1$
  - if $F(i+1,r)=0$ ($r=j_1 \ldots j_{k-1}$) then $F(i+1,r)=1$, $D(i,r)=1$ otherwise a "fatal failure" is reached.

An example of reconfiguration is given in fig. 1. Cells with R(i,j)=1 or D(i,j)=1 contain arrows pointing to the right or downwards, respectively, that represent deformation lines.

Each cell can require, at its inputs, signals coming from a small set of its neighbors, depending from the reconfiguration. As yet said in section 5, one possible implementation approach can be based upon adoption of input multiplexers: each multiplexer selects the right signal, thus activating the right spare path. Fig. 2 shows a possible array scheme, depicting processing elements, input multiplexers and spare paths.

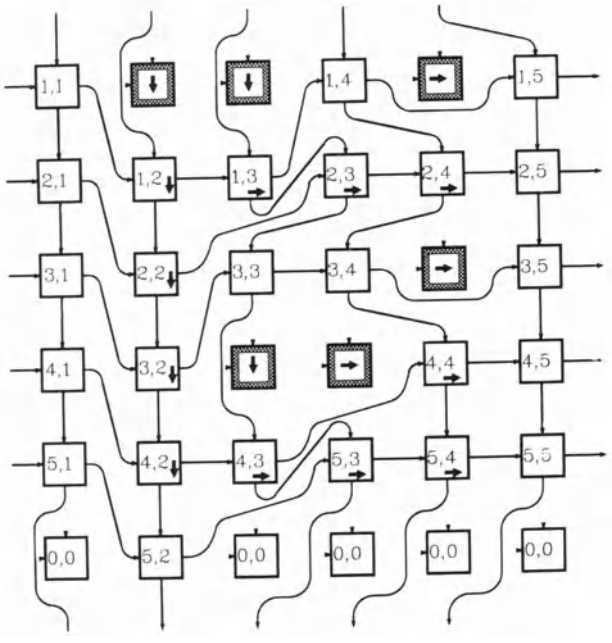


Fig. 1: Example of VLSI reconfiguration (Simple Fault- Stealing, Fixed Choice Algorithm).

Fig. 3 shows an implementation solution based upon buses and switches, referring to the same reconfiguration algorithm.

This algorithm shows better statistical survival performances if compared to the previous simple one, having only a column of spares. A 20*20 array has 50% probability of surviving to 19 faults.

In [13] more complex and robust algorithms are described, that present 50% probability of surviving to 37 faults in the same 20*20 array.

7. SELF-RECONFIGURATION OF WSI MESHES

In this section, we follow an approach to self- reconfiguration that has been introduced by the same authors ([14], [15], [16], [17], and [11]) and is capable of overcoming also large clustered distributions of faults, typical of WSI. First of all, "neighboring sets" must be fixed, i.e. it is necessary to state the number of neighbors that can be connected to every cell in order to prolong rows and columns of the array. If the right (row) neighbor can be chosen among x cells, and the lower (column) neighbor among y cells, the two neighboring sets are identified by the symbol /x*y/.

Let now consider the first class of reconfiguration algorithms, derived from methods treated in [14], [15] and [16], that adopt space redundancy. These algorithms reconfigure the array by identifying locally at each cell how to connect the cell, i. e. by choosing, inside the two neighboring sets for rows and columns, the two cells that must be connected to prolong the row and the column.

Locally, each cell manages "Request" and "Acknowledge" signals: starting from two edges of the physical array, requests to prolong rows

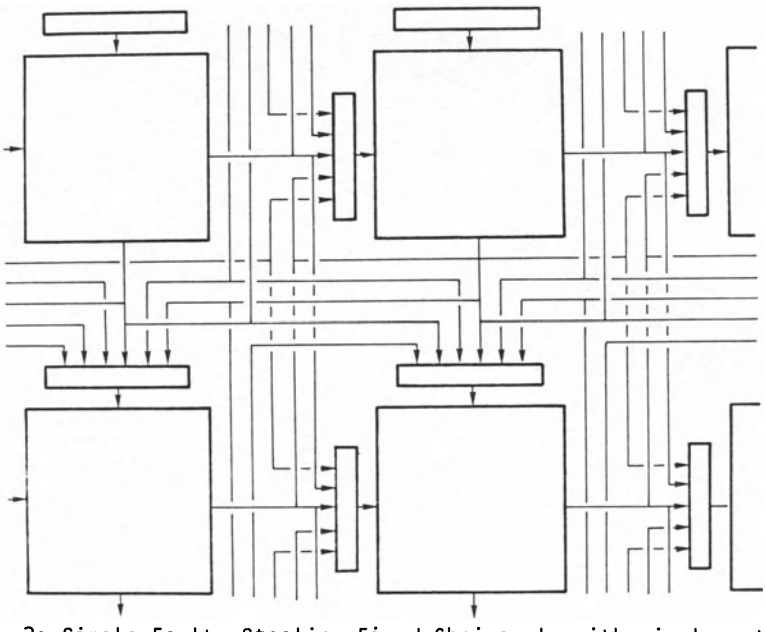


Fig. 2: Simple Fault- Stealing Fixed Choice algorithm implemented by input multiplexers and spare paths.

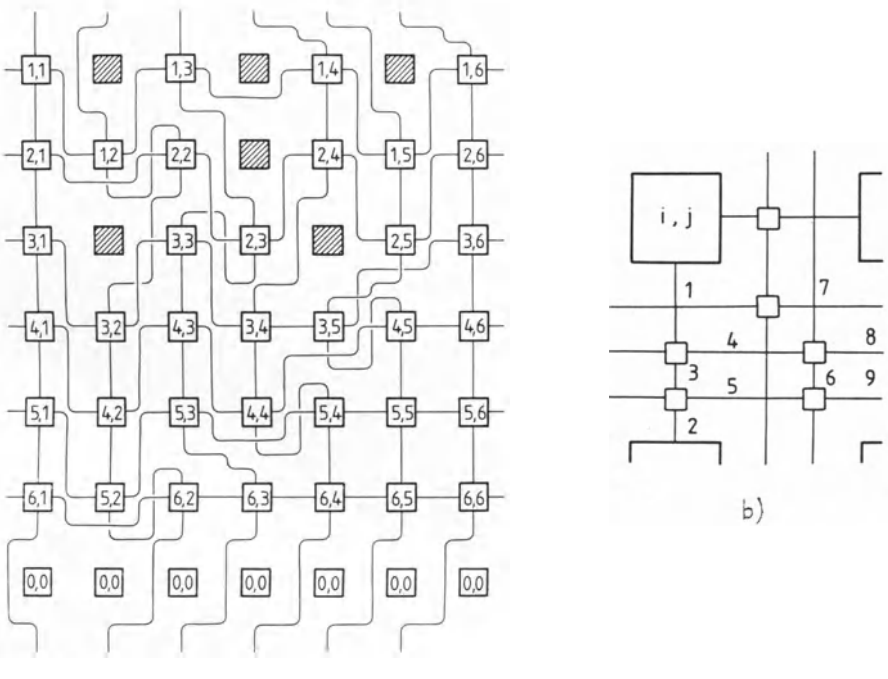


Fig. 3: The same algorithm of fig. 1 and 2 implemented by buses and switches. Fig. 3a shows an example of reconfiguration; fig. 3b shows position of switches interconnecting horizontal and vertical segments of buses.

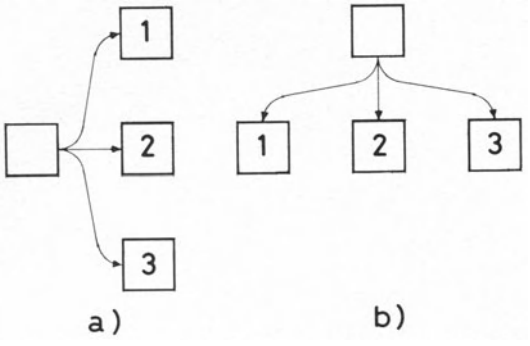


Fig. 4: The /3*3/ neighboring set.

and columns are then propagated inside the array. For example, when a working cell receives a request to prolong a row, it contemporarily passes requests to all the cells in its row neighboring set. A requested cell accepts the request only if it finds one of its working neighboring cells that can again prolong the row. The accepting cell sends an acknowledge signal back to the requesting cell.

Fixed priorities are associated to these signals: fig. 4 shows an example of /3*3/ neighboring sets, and figures inside cells define priorities both for acknowledges and for requests. Consider e.g. fig. 4a, showing row neighboring set: request signals are sent to cells 1, 2 and 3

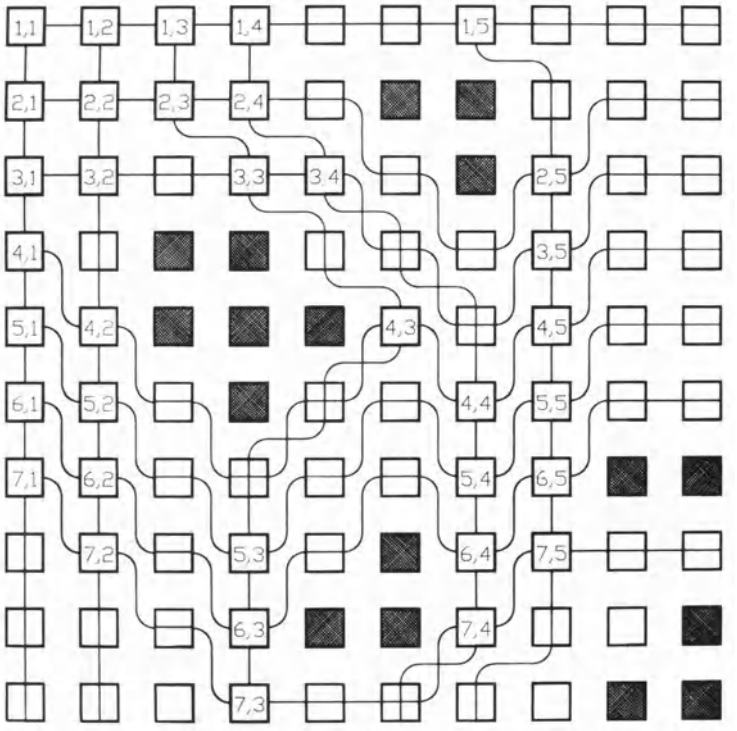


Fig. 5: Reconfiguration example with the /3*3/ algorithm.

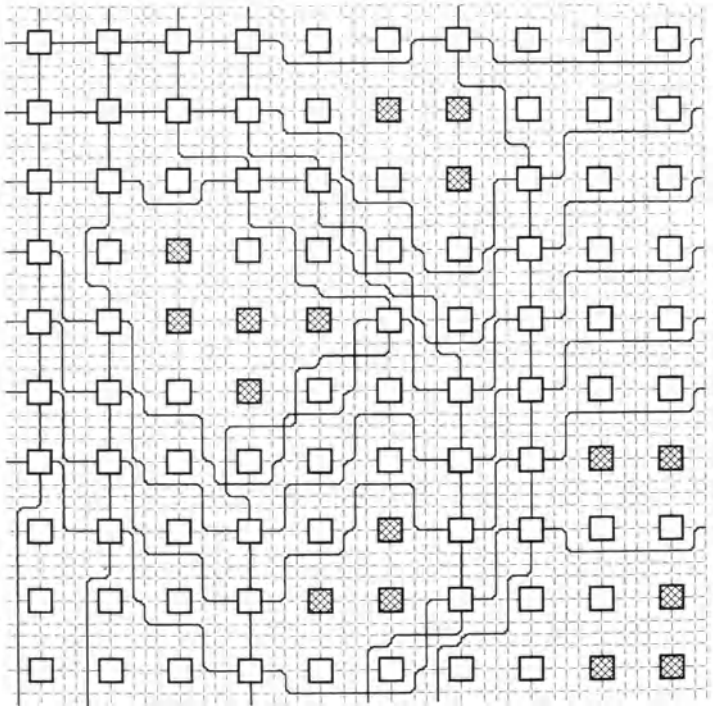


Fig. 6: The same reconfiguration problem of fig. 5, solved adopting buses and switches.

contemporarily. When an acknowledge signal comes back from cell i, it cancels all Requests associated with cell numbers higher than i. Thus, only the accepted request associated with the lower number is maintained.

Contemporary request signals might arrive to a cell coming from different cells: in this case, the same numbers of fig. 4a can be associated to requests (i.e. 1 associated to requests coming from the down-left neighbor, 2 associated to request coming from the left neighbor, and 3 to requests coming from upper-left neighbor). Then, only the higher-number request must be accepted.

Fig. 5 shows an example of reconfiguration based upon the /3*3/ neighboring sets. In this figure, inside cells are shown indices of position in the working array; relay cells are shown without indices.

These algorithms can be implemented by logic networks in all the ways recalled in Section 5, adopting spare paths and input multiplexers, or buses and switches. Fig. 6 shows an example, based upon switches and buses.

The request- acknowledge approach has been evaluated by simulating a relevant number of reconfiguration problems for the case of an array of 20*20 cells, injecting faults inside the cells. The reconfiguration capacity is measured by inspecting the dimensions of the largest working array that is obtained by applying the reconfiguration algorithm under test. In this way, harvesting is determined, i.e. the number of cells that constitute working array (excluding passing elements, not used working cells, and faulty cells).

In fig. 7, harvesting is shown as a function of the total number of injected faults, for the case of /3*3/ neighboring sets. A "+" identifies the harvesting for an array hit by clustered faults, a "o" does the same for random faults.

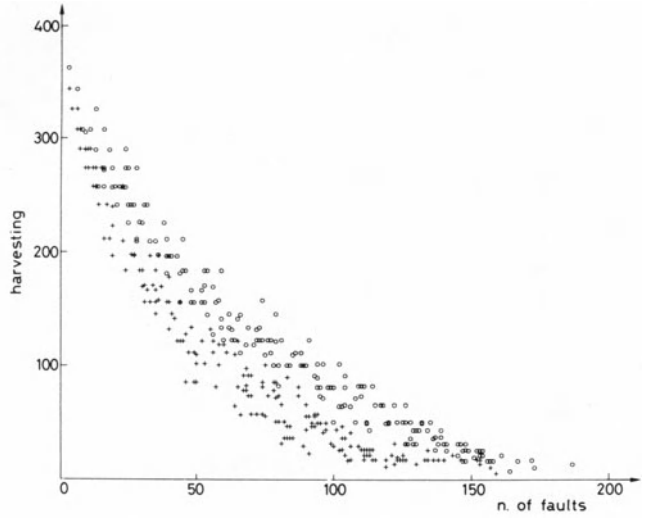


Fig. 7: Harvesting of /3*3/ in a 20*20 physical array.

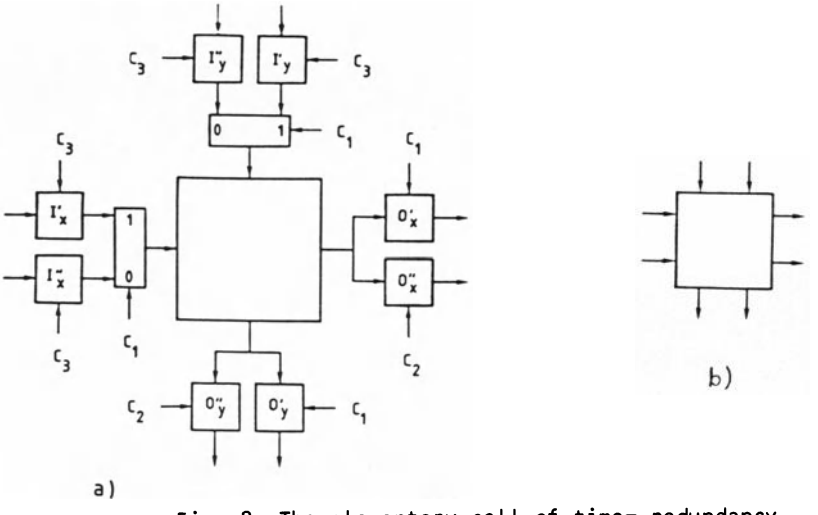


Fig. 8: The elementary cell of time- redundancy.

In [18] different algorithms of this class are compared, and it is shown that even algorithms corresponding to small neighboring sets give reasonably good harvesting. They also require low circuit area overheads for spare paths; this agrees with [5], that shows that faults hitting added circuit areas rapidly decrease global reliability when added area increases.

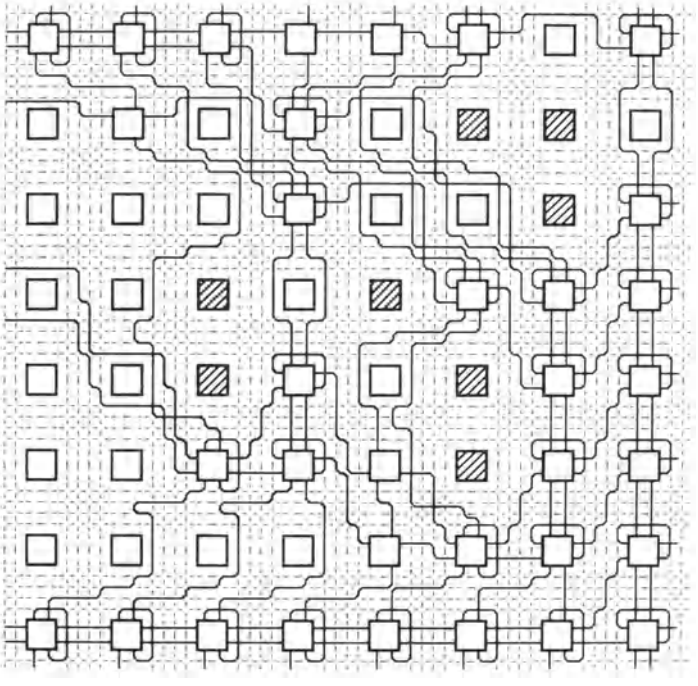


Fig. 9: Reconfiguration example by means of time- redundant /3*3/.

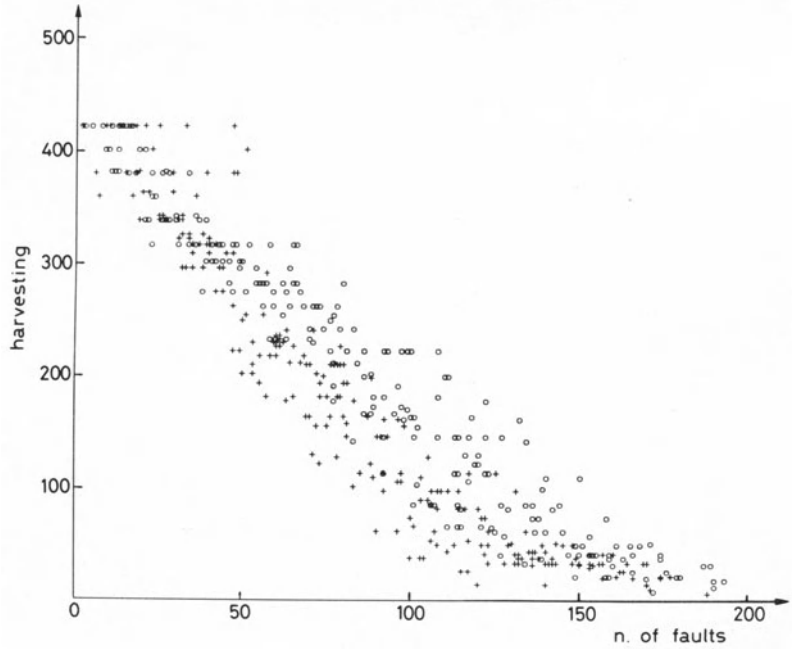


Fig. 10: Harvesting of time- redundant /3*3/ in a 20*20 array.

Time redundancy is adopted in the second group of algorithms for WSI meshes discussed in this section. The array operates in (at least) two alternating and synchronous phases: in the first phase each working cell executes its nominal functions; in the second phase all cells are idle and can execute the functions of faulty neighbors. This approach is particularly suited to the cases where very low amounts of spare cells can be adopted.

In [17] it has been shown that also time redundancy can be applied to reconfiguration of WSI arrays by means of request and acknowledge signals previously considered, provided that each physical cell corresponds to two "virtual" cells (one for each working phase) so that a "virtual" array can be reconfigured. It is sufficient that the cell has the structure depicted in fig. 8: input horizontal and vertical signals are selected and synchronized to the alternating phases by means of the multiplexers and latches shown in figure.

An example of reconfiguration is shown in fig. 9 where cells are drawn following the simplified rules of fig. 8b. Each physical cell is connected by spare links to 3 vertical neighbors and to 3 horizontal neighbors. Again, also in this case the reconfiguration might be implemented by means of switched buses, as shown in [18].

Harvesting properties for this case are shown in fig. 10, obtained by simulating random and clustered faults in physical cells.

8. EXTENSIONS TO IMAGE-PROCESSING ARRAYS

The analysis previously carried on refers to planar rectangular arrays of cells having two inputs and two outputs; they will be denoted as "wavefront-computation arrays". These arrays find application also in image processing, for particular functions, like DFT computation, but planar arrays for image processing are more often composed of cells receiving, through a multiplexer or a more complex combinatorial circuit inputs, the outputs coming from all the four neighbors (N, S, E, W) and passing its output to all the same four neighbors; this architecture will be denoted as "image array".

In this case, reconfiguration methods previously discussed can again be applied, changing the structure of hardware connections.

Let us consider a wavefront array: the following sets are defined:

- $S_W(i,j)$ is the set of cells that pass their horizontal output to the horizontal input of cell (i,j), through, e.g. the horizontal input multiplexer of (i,j);
- $S_N(i,j)$ is the set of cells that pass their vertical output to the vertical input of cell (i,j);
- $S_E(i,j)$ is the set of cells that receive its horizontal input from the horizontal output of cell (i,j);
- $S_S(i,j)$ is the set of cells that receive its vertical input from the vertical output of cell (i,j).

If the "simple fault stealing, fixed choice" algorithm is considered, it is:

$$S_N(i,j)=\{(i-1,j),(i,j-1),(i,j-2),(i+1,j-1),(i+1,j-2)\}$$

$$S_S(i,j)=\{(i+1,j),(i,j+1),(i,j+2),(i-1,j+1),(i-1,j+2)\}$$

$$S_W(i,j)=\{(i-2,j+1),(i-2,j),(i-1,j+1),(i-1,j),(i-1,j-1),(i,j-1)\}$$

$$S_E(i,j)=\{(i+2,j-1),(i+2,j),(i+1,j-1),(i+1,j),(i+1,j+1),(i,j+1)\}$$

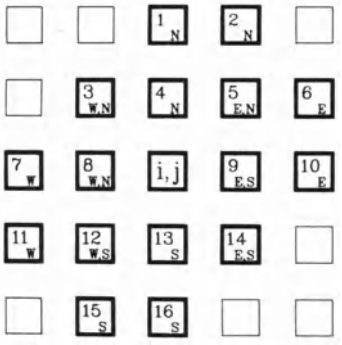


Fig. 11: Set of cells that can be chosen as N-E-S-W with Simple Fault Stealing algorithm

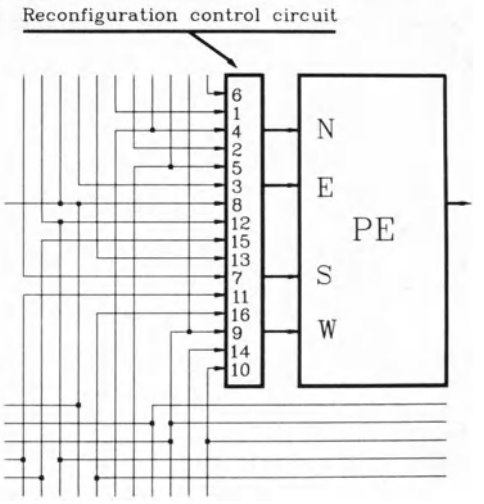


Fig. 12: elementary processing cell capable of choosing neighbors in the set of fig. 11

Consider now the "image array": the set S(i,j) of all cells whose output may be passed to the inputs of cell (i,j) under a reconfiguration due to the presence of some faulty cells, is

$$S(i,j) = S_N(i,j) \cup S_W(i,j) \cup S_S(i,j) \cup S_E(i,j)$$

If the "simple fault stealing, fixed choice" algorithm is considered, the set S(i,j) is presented in fig. 11; cell 14, as an example, may not be connected to cell (i,j) or it may be connected to (i,j) as a S-neighbor or as a E-neighbor depending on the reconfiguration.

The structure of an elementary PE is shown in fig. 12; the true PE receives its four inputs from a "Reconfiguration Control Circuit" (RCC); the outputs of the cells in S(i,j) are connected to the inputs of RCC through a vertical and an horizontal bus.

The reconfiguration control circuit can be constituted of four multiplexers selecting four cells from S(i,j); four registers store the control signals controlling the multiplexers. Fault-tolerance is then achieved through the proper setting of the four registers of each cell.

The previously discussed methods can also be extended to the case of three- dimensional (e.g. pyramidal) arrays. Extension to pyramids is complicated by the complex relationship and communications existing among cells of the array, and loss of communications locality when pyramids are projected onto flat surfaces.

From a geometric point of view, spare cells might be added in many ways to a pyramidal array. Cells might be added on the borders of each plane composing the pyramid. Extra planes also might be added, e. g. one at the bottom of the pyramid, possibly de-populated.

In practice, it is necessary and more important to account for how cells are distributed inside chips and printed circuit boards, because this can severely limit possibility of reconfiguration for necessity of introducing a high number of different redundant spare paths, since in pyramids each cell is linked to a large set of neighbors (e.g. to a father cell on the upper plane, to four brother cells on the same plane, and to four son cells on to lower plane)

In the simplest case, redundant cells can be inserted in a chip. If a VLSI chip contains 5 cells that constitute the smallest elementary pyramid, then e.g. one more cell can be added, and the elementary pyramid can be dynamically reconfigured through buses and switches, thus accepting the presence of one faulty cell for each chip. This method allows for self-reconfiguration of the pyramid, in case of chips containing reasonably small sub-pyramids.

When externally driven reconfiguration is sufficient, the solution can be found again in complex reconfiguration algorithms, based for example upon partitioning. It is then necessary to allow that inputs of a cell can be taken from outputs of a set of different neighboring cells, larger than in the cases seen for meshes. If planes are identified (even if not reconstructed) before than reconstructing the pyramid, this set must include cells that are candidate fathers, candidate brothers and candidate sons, and it would not be possible to mix them. Algorithms of this class are now being studied.

When static reconfiguration is sufficient, also reasonably good solutions can be found by means of complex, end- of- production static routing algorithms, based upon a regular structure composed by uniformly distributed cells, separated by wide routing channels.

9. CONCLUSIONS

In this paper, basic methods that can be applied to arrays of processing elements, in order to introduce in them fault- tolerance, have been considered.

Discussion has been focused on the case of arrays structurally similar to those used for image processing, even if, in general, reconfiguration methods tend to be defined in ways that are not strictly dependent from the envisioned applications of the arrays.

The main properties of reconfiguration algorithms for fault- tolerance, and their basic figures of merit, have been considered.

Algorithms that can be used to reconfigure automatically two-dimensions arrays (meshes) and that can be extended to pyramidal arrays, have been discussed in detail.

10. REFERENCES

[1] C.H.Stapper, "Modeling of Integrated Circuits Defect Sensitivities", IBM Journal of Research and Development, vol. 27, n. 6, pp. 549-557 (nov. 1983)

[2] W.R.Moore, "A Review of Fault- Tolerant Techniques for the Enhancement of Integrated Circuit Yield", Proceedings of the IEEE, Special issue on Fault-Tolerance for VLSI (may 1986)

[3] R.Negrini, M.G.Sami, N.Scarabottolo, "Policies for System-Level Diagnosis in a Nonhierarchical Distributed System", IEEE Trans. Reliab. (Oct. 1984)

[4] D.P.Siewiorek, R.S.Swarz, The Theory and Practice of Reliable System Design, Digital Press, Bedford (1982)

[5] R.M.Mangir, A.Avizienis, "Fault-tolerant design for VLSI: effect of interconnection requirements on yield improvement of VLSI design", IEEE Trans. Comp., vol. C31, n. 7, pp. 609-615 (July 1982)

[6] A.Antola, R.Negrini, M.G.Sami, N.Scarabottolo, "Transient Fault Management in Microprogrammed Units: a Software Recovery Approach", Proc. EUROMICRO '85, Bruxelles (Sept. 1985)

[7] K.E. Batcher, "Architecture of a Massively Parallel Processor", Proc. 7th Symp. on Computer Architecture, pp. 168-173, IEEE (May 1980)

[8] H.T.Kung, M.S.Lam, "Fault-tolerance and two-level pipelining in VLSI systolic arrays", MIT conf. on Advanced Research in VLSI (Jan. 1984)

[9] J.W.Greene, A. El Gamal, "Configuration of VLSI Arrays in the Presence of Defects", JACM, vol. 31, n. 4, pp. 694-717 (oct. 1984)

[10] T.Leighton, C.E.Leiserson, "A Survey of Algorithms for Integrating Wafer- Scale Systolic Arrays", Proc. IFIP Workshop on WSI, North-Holland, Grenoble (mar. 1986)

[11] R.Negrini, M.G.Sami, R.Stefanelli, "Fault Tolerance Techniques for Array Structures Used in Supercomputing", IEEE Computer Magazine (Feb. 1986)

[12] M.G.Sami - R.Stefanelli, "Reconfigurable architectures for VLSI processing arrays", Proceedings of the IEEE, Special issue on Fault-Tolerance for VLSI (may 1986)

[13] M.G.Sami, R.Stefanelli, "Fault-stealing: an approach to fault-tolerance of VLSI array structures", Proc. ICCAS 85, IEEE, Beijing (June 1985)

[14] R.Negrini, R.Stefanelli, "Reconfigurable architectures for fault-tolerant wafer- scale regular arrays", Int. Rep., Dept. of Electronics, Polytecnic of Milan, Milan (feb. 1984)

[15] R.Negrini, M.G.Sami, R.Stefanelli, "Fault- Tolerance Approaches for VLSI/WSI Arrays", Proc. CCC (Conf. on Computers and Communications), IEEE, Phoenix (1985)

[16] R.Negrini, R.Stefanelli, "Algorithms for self- reconfiguration of wafer- scale regular arrays", Proc. ICCAS, IEEE, Beijing (1985)

[17] R.Negrini, R.Stefanelli, "Time Redundancy in WSI Arrays of Processing Elements", Proc. 1st Int. Conf. on Supercomputing Systems, St. Petersburg (Dec. 1985)

[18] R.Negrini R.Stefanelli, "Comparative Evaluation of Space- and Time-Redundancy Approaches for WSI Processing Arrays", Proc. Workshop on WSI, North- Holland, Grenoble (mar. 1986)

NATO ASI Series F

Vol. 1: Issues in Acoustic Signal – Image Processing and Recognition. Edited by C. H. Chen. VIII, 333 pages. 1983.

Vol. 2: Image Sequence Processing and Dynamic Scene Analysis. Edited by T. S. Huang. IX, 749 pages. 1983.

Vol. 3: Electronic Systems Effectiveness and Life Cycle Costing. Edited by J. K. Skwirzynski. XVII, 732 pages. 1983.

Vol. 4: Pictorial Data Analysis. Edited by R. M. Haralick. VIII, 468 pages. 1983.

Vol. 5: International Calibration Study of Traffic Conflict Techniques. Edited by E. Asmussen. VII, 229 pages. 1984.

Vol. 6: Information Technology and the Computer Network. Edited by K. G. Beauchamp. VIII, 271 pages. 1984.

Vol. 7: High-Speed Computation. Edited by J. S. Kowalik. IX, 441 pages. 1984.

Vol. 8: Program Transformation and Programming Environments. Report on an Workshop directed by F. L. Bauer and H. Remus. Edited by P. Pepper. XIV, 378 pages. 1984.

Vol. 9: Computer Aided Analysis and Optimization of Mechanical System Dynamics. Edited by E. J. Haug. XXII, 700 pages. 1984.

Vol. 10: Simulation and Model-Based Methodologies: An Integrative View. Edited by T. I. Ören, B. P. Zeigler, M. S. Elzas. XIII, 651 pages. 1984.

Vol. 11: Robotics and Artificial Intelligence. Edited by M. Brady, L. A. Gerhardt, H. F. Davidson. XVII, 693 pages. 1984.

Vol. 12: Combinatorial Algorithms on Words. Edited by A. Apostolico, Z. Galil. VIII, 361 pages. 1985.

Vol. 13: Logics and Models of Concurrent Systems. Edited by K. R. Apt. VIII, 498 pages. 1985.

Vol. 14: Control Flow and Data Flow: Concepts of Distributed Programming. Edited by M. Broy. VIII, 525 pages. 1985.

Vol. 15: Computational Mathematical Programming. Edited by K. Schittkowski. VIII, 451 pages. 1985.

Vol. 16: New Systems and Architectures for Automatic Speech Recognition and Synthesis. Edited by R. De Mori, C.Y. Suen. XIII, 630 pages. 1985.

Vol. 17: Fundamental Algorithms for Computer Graphics. Edited by R. A. Earnshaw. XVI, 1042 pages. 1985.

Vol. 18: Computer Architectures for Spatially Distributed Data. Edited by H. Freeman and G. G. Pieroni. VIII, 391 pages. 1985.

Vol. 19: Pictorial Information Systems in Medicine. Edited by K. H. Höhne. XII, 525 pages. 1986.

Vol. 20: Disordered Systems and Biological Organization. Edited by E. Bienenstock, F. Fogelman Soulié, G. Weisbuch. XXI, 405 pages. 1986.

Vol. 21: Intelligent Decision Support in Process Environments. Edited by E. Hollnagel, G. Mancini, D. D. Woods. XV, 524 pages. 1986.

Vol. 22: Software System Design Methods. The Challenge of Advanced Computing Technology. Edited by J. K. Skwirzynski. XIII, 747 pages. 1986.

NATO ASI Series F

Vol. 23: Designing Computer-Based Learning Materials. Edited by H. Weinstock and A. Bork. IX, 285 pages. 1986.

Vol. 24: Database Machines. Modern Trends and Applications. Edited by A.K. Sood and A.H. Qureshi. VIII, 570 pages. 1986.

Vol. 25: Pyramidal Systems for Computer Vision. Edited by V. Cantoni and S. Levialdi. VIII, 392 pages. 1986.