REFERENCES

[1] Z. Kulpa, "Area and perimeter measurement of blobs in discrete binary pictures," *Comput. Graphics Image Processing*, vol. 6, pp. 434-454, 1977.

[2] R. C. Gonzalez and Safabakhsh, "Computer vision techniques for industrial applications and robot control," *IEEE Computer*, vol. C-15, pp. 17-32, Dec. 1982.

[3] A. Rosenfeld, "Digital straight line segments," *IEEE Trans. Comput.*, vol. C-23, pp. 1264-1269, 1974.

[4] F. C. A. Groen and P. W. Verbeek, "Freeman-code probabilities of object boundary quantized contours," *Comput. Graphics Image Processing*, vol. 7, pp. 391-402, 1978.

[5] H. Freeman, "Boundary encoding and processing," in *Picture Processing and Psychopictorics*, B. S. Lipkin and A. Rosenfeld, Eds. New York: Academic, 1970, pp. 241-266.

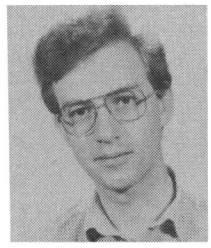
[6] A. M. Vossepoel and A. W. M. Smeulders, "Vector code probability and metrication error in the representation of straight lines of finite length," *Comput. Graphics Image Processing*, vol. 20, pp. 347-364, 1982.

[7] G. H. Hardy and E. M. Wright, *An Introduction to the Theory of Numbers*, 5th ed. Oxford: Clarendon, 1979, ch. V.

[8] A. Rosenfeld and A. C. Kak, *Digital Picture Processing*, vol. 2, 2nd ed. New York: Academic, 1982, pp. 121-126.

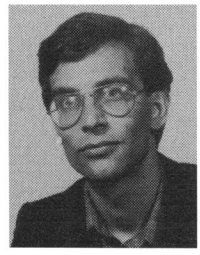
[9] L. Dorst and A. W. M. Smeulders, "The estimation of parameters of digital straight line segments," in *Proc. 6th ICPR*, München, 1982.

[10] A. W. M. Smeulders and L. Dorst, TH Delft, Internal Rep. 1983.



Leo Dorst was born in Rotterdam, The Netherlands, in 1958. He received the "Ingenieur" degree (M.Sc.) in applied physics from Delft University of Technology, Delft, The Netherlands, in 1982.

He is working there now using digital image processing for quantitative interferometry, a project supported by the Dutch FOM Foundation. His research interests include digital image processing, pattern recognition and, particularly, the accuracy of measurements in images.



Arnold W. M. Smeulders was born in The Hague, The Netherlands, in 1952. He received a degree in applied physics from Delft University of Technology, Delft, The Netherlands, in 1977, and the Ph.D. degree from Leyden State University in 1983.

Before completing his Ph.D. dissertation on pattern analysis of cervical specimens, he worked on pattern recognition of chromosome images at Delft University of Technology. Presently, he is with the Department of Pathology and Medical Informatics at the Free University, Amsterdam, The Netherlands. His research interests include digital image measurement, medical image understanding, and decision making.

Multiprocessor Pyramid Architectures for Bottom-Up Image Analysis

NARENDRA AHUJA, MEMBER, IEEE, AND SOWMITRI SWAMY

Abstract—**This paper describes three hierarchical organizations of small processors for bottom-up image analysis: pyramids, interleaved pyramids, and pyramid trees. Progressively lower levels in the hierarchies process image windows of decreasing size. Bottom-up analysis is made feasible by transmitting up the levels quadrant borders and border-related information that captures quadrant interaction of interest for a given computation. The operation of the pyramid is illustrated by examples of standard algorithms for interior-based computations (e.g., area) and border-based computations of local properties (e.g., perimeter). A connected component counting algorithm is outlined that illustrates the role of border-related information in representing quadrant interaction. Interleaved pyramids are obtained by sharing processors among several pyramids. They increase processor utilization and throughput rate at the cost of increased hardware. Trees of shallow interleaved pyramids, called pyramid trees, are introduced to reduce the hardware requirements of large interleaved pyramids at the expense of increased processing time, without sacrificing processor utilization. The three organizations are compared with respect to several performance measures.**

Index Terms—**Divide-and-conquer, image analysis, image decomposition, interleaving, parallel processing, performance evaluation, pipelining, pyramid architectures.**

I. INTRODUCTION

THIS paper explores the use of hierarchical organization of processors to perform strictly bottom-up computations. Three architectures are described: pyramids, interleaved pyramids, and pyramid trees. These architectures perform computations that result in a small number of output bits (small compared to the number of bits necessary to represent the entire image). The architectures are thus intended to compute image properties or to perform image analysis. They are not suitable for performing image transformations, such as seg-

Manuscript received February 23, 1983; revised July 18, 1983. This work was supported by the Joint Services Electronics Program, U.S. Army, Navy and Air Force, under Contract N00014-79-C-0424.

N. Ahuja is with the Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, Urbana, IL 61801.

S. Swamy was with the Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, Urbana, IL 61801. He is now with the Department of Computer Science, Gould Research Center, Rolling Meadows, IL 60008.

0162-8828/84/0700-0463$01.00

mentation or enhancement, which provide a whole image as output.

The central feature of the architectures described is a hierarchy imposed on the image by a recursive square decomposition of the image. This hierarchy has formed the basis of various pyramid approaches to be reviewed later in this section and the quadtree representation of images. To obtain its quadtree, the image is overlaid with a sequence of increasingly fine tessellations that define a recursive embedding of quadrants and thus a hierarchy over image windows. The hierarchy is described by a tree whose root node is associated with the entire image. Each node in the tree represents a square window. A nonleaf node has four children. Each child node is associated with a quadrant of the parent window. Leaf nodes correspond to pixels or to windows of the smallest size.

Consider a set of images and the set of nodes in their tree representations. Different mappings from the nodes (windows) of possibly more than one tree (image) to a set of processors define different processor hierarchies. The processors perform computations on their corresponding windows. A single tree (image) is processed a level at a time starting at the bottom and progressing up the levels. The results of computations on the images are obtained at the processors corresponding to the root nodes. The terms node and processor will be used interchangeably. A nonleaf node receives the results of computation from its children nodes and combines them, making use of the way in which the children quadrants interact, to obtain the result for its own window. Information about quadrant interaction is necessary to make strictly bottom-up computation possible. Clearly, it would be desirable to minimize the volume of such information flow up the hierarchy. For such simple computations as area, no information about quadrant interaction is necessary. For more involved computations such as perimeter, quadrant borders completely specify quadrant interaction. For complex computations such as connected component counting, connectivity properties of quadrant interiors consitute quadrant interaction and can be represented by borders and certain border related structures (Section III-B-2). A variety of image analysis tasks can be carried out efficiently on such hierarchies. Bottom-up algorithms can be adapted from literature to perform simple computations such as area and perimeter. More complex computations that can be implemented efficiently include connected component counting and transformation of point sets into graphs.

The choice of the mapping, possibly many to one, from the tree nodes to processors determines the different architectures discussed in this paper. The first choice assigns one processor to each node in the tree, yielding the standard multiprocessor pyramid organization proposed by Uhr in 1971 [1] (see also [2]-[4]) and called "recognition cone." A similar organization, called cellular pyramid acceptor, is also considered by Dyer [5]. There, processors are organized as a complete quadtree with additional interconnections between neighboring processors at the same level. Each node has a finite set of states. A node changes its state depending upon its current state and the states of its parent, children and siblings, and in parallel with all other nodes. A special case of bottom-up cellular pyramid acceptor has also been considered [6] where each node changes its state based upon its current state and the states of its children. The interconnections among the processors in the bottom-up cellular pyramids and in the first hierarchy discussed in this paper are similar. However, each nonleaf node in the latter case receives from its children the results of their computations and explicit information about quadrant interaction which cannot be naturally described in terms of states and state transitions. A small general-purpose processor is assumed at each node. An algorithm for counting the number of connected components is outlined that highlights the need for a general processor and complex border related information from quadrants, and distinguishes the use of the first hierarchy from the bottom-up pyramids described in [6]. In [7] Dyer considers a pyramid machine where each node has a small processor. He discusses the node interconnections and chip layout aspects with respect to VLSI design. This paper addresses the dataflow aspects and implementation of bottom-up algorithms.

The remaining two choices of the node to processor mappings are many to one. Both can be viewed as hierarchies using different interconnections among pyramids. The first of these two choices involves interleaving of pyramids where processors at a single level of the hierarchy may simultaneously process windows from several images. Interleaved pyramids achieve high throughput at the cost of increased hardware. The second choice involves a hierarchy of pyramids interconnected as a tree, with each pyramid being capable of processing a subimage. A large image is divided into sets of subimages which are processed in parallel by the nodes of the pyramid tree. Pyramid trees reduce input bandwidth and hardware costs as compared to the interleaved pyramids at the expense of increased computation time.

The pyramid organization of processing elements has received the attention of several researchers as noted earlier. Rosenfeld [8] presents an enlightening treatment of cellular pyramids, cellular arrays, other related machines, and their performance with respect to commonly used image operations. Some of the work is motivated by the quadtree representation [9], [10] of images that use recursive decomposition [11] of an image for the purpose of compact representation rather than multiprocessing. Pyramids have also been used to obtain a hierarchical representation of an image at several resolutions, not necessarily to be processed by more than one processor; see Kelly [12], Levine [13], Hanson and Riseman [2], Price and Reddy [14], Rosenfeld [9], Tanimoto [15], and Tanimoto and Pavlidis [16]. The coarser resolutions are obtained by, for example, applying averaging operators to the pixels of a window. The analysis at higher levels is approximate but quick, and is used to "plan" the more detailed analysis at lower levels. Multiresolution pyramids may have overlap between the windows of adjacent nodes [17]. Multiresolution pyramids to represent edges at different resolutions [18] have also been investigated.

A hierarchical architecture called active quadtree networks, which is an adaptation of the quadtree representation to multiprocessing, has been described by Dubitzki *et al.* [19]. They assign a processor to each block in the quadtree representation of the image. The number of and interconnections among processors are thus image dependent.

Among the nonhierarchical architectures, a common approach

is to assign a processor to each pixel or to each neighborhood of pixels with local interconnections among processors. Examples of such mesh architectures are MPP [20], CLIP4 [21], and DAP [22]. Other architectures establish dynamic interconnections among processors (ZMOB [23], PASM [24]).

Section II describes the architectural details of the multiprocessor pyramid. Section III briefly describes standard algorithms for interior-based computations (e.g., area) and border-based computations (e.g., perimeter). Section III further describes a connected component counting algorithm to illustrate the role of border-based information in making bottom-up analysis feasible. The computational complexity and memory requirements are also discussed. Interleaved pyramids and pyramid trees are described in Sections IV and V as architectures for achieving increased processor utilization. Section VI describes performance measures that are used to evaluate the architectures. Section VII presents concluding remarks. The Appendix gives detailed derivations of the performance measures for the pyramid tree.

II. The Pyramid Architecture

This section describes the pyramid architecture, the first and the most basic of the three hierarchical organizations of processing elements to be discussed in this paper.

A. Processor Hierarchy

Assume that the windows of the finest tessellation contain a single pixel. If such is not the case, only trivial modifications to the description need to be made. The pyramid architecture is reviewed below using a notation similar to that used by Burt *et al.* [17]. Consider a $2^L \times 2^L$ image. A pyramid to process this image is a layered arrangement of $L+1$ square arrays of processing elements (PE) $A(0), A(1), \cdots, A(L)$. The bottom array is at level 0 and is of size $2^L \times 2^L$. Arrays at higher levels have their dimensions reduced by half from level to level. Thus, $A(1)$ measures $2^{L-1} \times 2^{L-1}$ and any $A(l)$, $0 \leqslant l \leqslant L$, measures $2^{L-l} \times 2^{L-l}$. The number of PE's is reduced by a factor of four from level to level. The number of PE's in $A(l)$, $0 \leqslant l \leqslant L$, is $2^{2(L-l)}$. $A(L)$ has a single PE which is the apex of the pyramid (Fig. 1).

Each PE in the pyramid is indexed by a triple (i, j, l), where l is its array level and i and j are its row and column numbers within that array. Rows and columns in an array $A(l)$ are numbered $0, 1, \cdots, 2^{L-l}-1$, from bottom to top and from left to right, respectively. The interconnections among various PE's in the pyramid are given by their i, j, and l coordinates, viewing the pyramid as a three-dimensional structure. Each PE (i, j, l) has a position $(x(i, l), y(j, l))$ relative to the image. Let the pixels be placed at the centers of the square windows of the finest tessellation. If the origin of the x, y plane is assumed to coincide with the bottom-left corner of the Euclidean image, then the continuous coordinates of the bottom-left pixel are given by

$$x(0, 0) = y(0, 0) = \tfrac{1}{2}$$

where the north and east directions are labeled as positive x and y axes, respectively. Then

$$x(i, 0) = i + \tfrac{1}{2}, \quad 0 \leqslant i \leqslant 2^L - 1$$
$$y(j, 0) = j + \tfrac{1}{2}, \quad 0 \leqslant j \leqslant 2^L - 1.$$

At any level $l > 0$

$$x(0, l) = y(0, l) = 2^{l-1}$$

and

$$x(i, l) = x(0, l) + i2^l = 2^{l-1}(1 + 2i), \quad 0 < i \leqslant 2^{L-l} - 1$$
$$y(j, l) = 2^{l-1}(1 + 2j), \quad 0 < j \leqslant 2^{L-l} - 1.$$

Each PE indexed (i, j, l) corresponds to the $2^l \times 2^l$ window centered at $(x(i, l), y(j, l))$. The total length of the border of this window is $4(2^l - 1)$ since each corner pixel is shared by two edges. The PE's corresponding to the quadrants of (i, j, l)'s window constitute (i, j, l)'s children at level $(l-1)$. The children PE's have indexes $(2i+1, 2j, l-1)$, $(2i+1, 2j+1, l-1)$, $(2i, 2j+1, l-1)$ and $(2i, 2j, l-1)$, respectively, (Fig. 2). A PE at level $l = 0$ corresponds to a single pixel which also constitutes its four border segments.

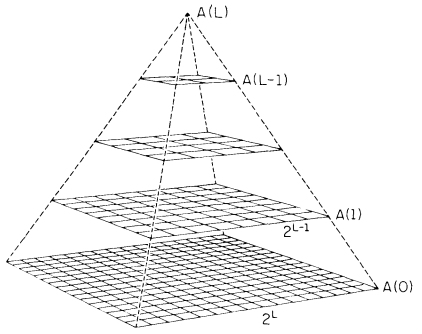


Fig. 1. Arrays in a $(L+1)$-level pyramid that can process a $2^L \times 2^L$ image.

B. Dataflow

As pointed out in Section I, the architecture to be described performs bottom-up computations on the image. In order to compute a given property at a window, its PE must receive data from its children. The data transmitted by a PE to its parent consist of the results of computation performed by the PE on its corresponding window, and border-related information. The latter is used by the parent PE to determine the interaction among its window's quadrants. In all three architectures discussed in this paper, the image data is loaded into the PE's at the base of the hierarchy in a manner determined by the architecture. Data paths are provided between each PE and its four children along which it receives, in parallel, data from its children. The hierarchical manner of computation necessitates limited information flow bottom to top; otherwise the data transmission time may dominate the overall computation time.

Now given a window, the structural relationships among its quadrants must depend upon their adjacent borders. Therefore, the quadrant interaction necessary for computing various properties may be expressed by borders and the relationships of quadrant interiors to their borders. For example, borders are required for perimeter computation (Section III-A), and con-

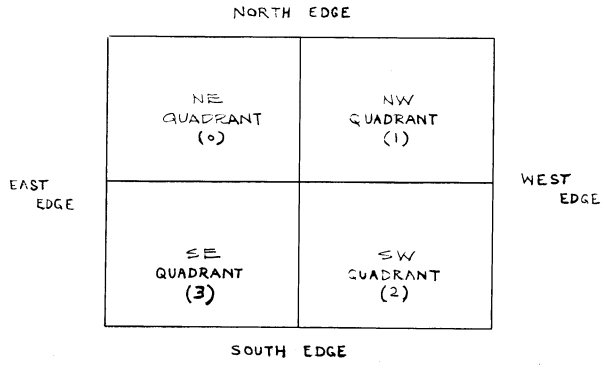


Fig. 2. The window corresponding to a node (i, j, l) and its four quadrants.

nectedness properties of border runs must be known to count number of connected components (Section III-B). The total volume of data to be transmitted from a PE to its parent is proportional to the length of the borders of its corresponding window–$0(n)$ for an $n \times n$ window. It is assumed that the data transmission rate is fixed for all data paths between PE's at adjacent levels. Thus, data transmission time per PE corresponding to an $n \times n$ window is $0(n)$. The total volume of data transmitted per PE doubles with level. However, since the number of PE's decreases by a factor of four per level, the overall dataflow up the pyramid halves with level.

C. Processor Element Hardware

The PE's are small general-purpose processors with identical CPU's and variable memory size. This is in contrast with the finite-state machine model of a processor used in bottom-up cellular pyramid acceptors, as pointed out in Section I. PE's at different levels of the pyramid execute the same set of instructions on windows of different sizes, and hence on different data. The execution of most algorithms is data dependent. This suggests that the PE's operate in a MIMD mode and that the instruction and data memories are segregated. The total size of the data memory for a PE at level k is $0(2^k)$. The data word size at all levels is fixed and is sufficiently large to hold the results ($0(\log_2 N)$ bits) for the largest $(N \times N)$ images of interest.[1]

III. Image Analysis

This section considers several image analysis algorithms for the pyramid architecture described in Section II. These algorithms are also valid for the interleaved pyramids and pyramid trees discussed in Sections IV and V. Binary images are considered where 0 and 1 denote the (white) background and (black) foreground, respectively. The algorithms are divided into two classes: interior-based algorithms, or those whose computations on a window involve only the results on the quadrants; and border-based algorithms, or those whose computations involve both results of the computations on its quadrants, and their borders. The chosen classes represent two major types of commonly performed image computations; each type has a different time complexity. The interior-based and simple border-based algorithms (e.g., local property counting), outlined below are straightforward adaptations of algorithms that have appeared in the literature. The connected component counting algorithm described is new, but details are skipped to maintain the focus of attention on the architectures.

A. Interior-Based Algorithms

These algorithms compute such properties of images that depend upon numbers and coordinates of the black pixels, but not on the relative positions of black and white pixels. Bottom-up computation of total black area and centroid in quadtree networks outlined in [19] can be used for pyramids. Each leaf of the pyramid is assigned an area measure of 0 or 1, corresponding to its color. Each nonleaf node then computes its area measure by adding the areas computed by its four children nodes. The area computed at the root is the total black area in the image. To compute the centroid of an image, the southwest corner pixel in any window is given the coordinates $(\frac{1}{2}, \frac{1}{2})$. Each black leaf is given the centroid coordinates of $(\frac{1}{2}, \frac{1}{2})$. The centroid at an interior node is computed as an area-weighted mean of the centroid values received from the four children nodes, with coordinates reexpressed with respect to the origin of the parent window.

B. Border-Based Algorithms

The properties computed by algorithms described in this section are determined by the number of black pixels and their neighborhood configurations. The neighborhood information is accumulated by processing border segments of adjacent quadrants of increasing size, unlike in quadtrees [9] and quadtree networks [19], where such information is extracted from blocks and their neighbors. Examples of border-based properties include local properties such as perimeter and genus, and the number of connected components that involves global connectivity among black pixels. Thus, additional information about which pixels along the borders of a quadrant are connected via black pixels in the quadrant's interior must be carried up the pyramid, along with the borders themselves. In this sense, the algorithms for such properties are more complex.

1) *Local Property Counting:* These algorithms are based upon the frequencies of various local configurations of 1's and 0's. These frequencies are computed from the adjacent border segments of quadrants of increasing sizes as computation progresses up the pyramid. Two examples of such computations are perimeter and genus.

The total perimeter of an image may be defined as the number of black-white pairs of neighbors in the image [25]. Thus, leaves (pixels) are assigned a perimeter value of 0. The perimeter at an interior node has two components: 1) that due to the black-white pairs of pixels within the node's quadrants, which is the sum of the perimeter values assigned to its children nodes, and 2) that due to the black-white pairs of pixels belonging to adjacent quadrants, which must be computed from the pairs of border segments of quadrants which are adjacent in the image. Computation at the root yields the total perimeter value for the image.

Genus of an image is defined as the number of components minus the number of holes. It can be computed from the histogram of the image defined over distinct classes of configurations of its 2×2 neighborhoods [25]. Each node carries

[1] Here and in the rest of the paper, the logarithm notation $\log x$ will denote $\log_2 x$ for simplicity.

the histogram for its window, which is initialized to a vector of zeros for leaves. The genus at a node is computed as a linear sum of the histogram bin counts. An interior node at level $l > 0$ contains $(2^l - 1)^2$, 2×2 neighborhoods formed by 1) the interiors of its four quadrants, each contributing $((2^l - 1)^2 - 1)/4$ neighborhoods, 2) four pairs of adjacent quadrant border segments, each pair contributing $(2^{l-1} - 1)$ neighborhoods, and 3) interior corner pixels of the quadrants, contributing the central neighborhood. The histogram for the node is obtained by computing the histograms of 2) and 3) and adding to them the four already known histograms for 1). Dyer [26] gives a genus algorithm for quadtrees that uses a similar approach. However, as mentioned earlier, the neighborhood information is gathered differently in quadtrees than in pyramids.

2) *Connected Component Counting:* Connected components of a binary image will refer to the black regions in the image. This section sketches an algorithm for bottom-up computation of the number of connected components. The details of the algorithm are tedious [28], and are skipped here to avoid diverting attention away from architectural issues.

Each node carries 1) a count of the number of components within its window that do not reach its border, and 2) for each component in the window that touches the border of that window, a doubly linked list linking the component's border intercepts (runs) in the same order as they are encountered along a border scan (Fig. 3). Leaves are assigned a component count of 0. The linked list of a leaf node is empty if it is white, and contains exactly one black run of unit length if the leaf is black. Each interior node receives from its children their counts and linked lists, and obtains its own count and linked list in three steps. First, the NW and NE quadrants are merged to obtain the count and linked lists for the northern half of the window. The same procedure is then repeated on the SW and SE quadrants. Finally, a third application of the procedure merges the northern and southern halves. Following is a brief description of the merge procedure illustrated for the NW and NE quadrants.

First, the adjacent border segments of the NW and NE quadrants are logically ORed to get merged runs along a common border. The merged runs represent components lying in the NW and NE quadrants that are adjacent. A linked list is then obtained to represent the corresponding connectivity among runs along the common border. Next, a count is obtained of those merged components that do not reach the external border segments (i.e., the border of the merged window). To do this, the common border is scanned in one direction, say from south to north, and congruence classes of runs are identified. Each class consists of all the runs belonging to a single component. Each class is examined for external links. If none of the runs is linked to runs along the external border segments, the class represents a component that does not reach the border of the merged window. For each such component found, the desired count is incremented by 1. Finally, a linked list is obtained corresponding to the remaining components for the new window formed by merging NW and NE quadrants. This is done by performing a scan (say south to north) of each congruence class of runs along the common border. A sequence of successive runs along a chain having links to external border segments is identified. For each scan, the nearest runs, if any,

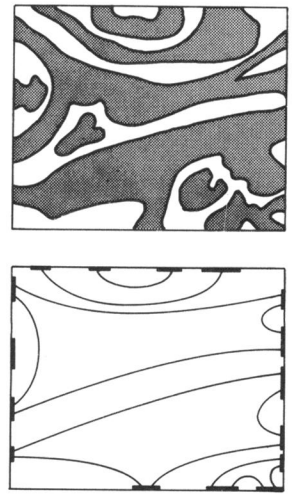


Fig. 3. Components in an image window and the corresponding set of doubly linked lists. Each list links successive border intercepts of a single component that reaches the window border. Links are shown by arcs joining pairs of border runs.

on the external border segments connected through runs on the common border directly linked. This gives a set of linked lists for the northern half. Over all congruence classes, the total number of (run) accesses is proportional to the number of runs in NW and NE quadrants [28]. The total number of accesses and the overall computation time of the algorithm is linear in the length of the border segments of the NW and NE quadrants.

C. Complexity Issues

This section gives estimates of the random access memory and computation time required by the pyramid to execute the algorithms described in Section III-A and B.

1) *Memory Requirements:* The total amount of local random access memory required by all the PE's depends on the type of computations to be performed. The interior-based algorithms do not involve any crossborder computations. Consequently, the amount of memory required at a PE at any level reflects the requirements of a fixed number of arithmetic operations on numbers representing area, centroid, etc. Since a fixed memory word length of $0(\log N)$ bits is used in PE's at all levels, the total memory required by each PE for the interior-based algorithms is $0(1)$ words.

The local property counting algorithms perform Boolean operations on crossborder bit pairs while the connected component counting algorithm generates and processes linked lists representing run and component connectivities. Let $C_m \cdot n$ be the memory required to store border information for an $n \times n$ window; the constant C_m does not depend on the window size, but depends on the algorithm. Since the memory required for arithmetic operations is small [0(1)] and border information is restricted to $0(n)$, the above two algorithms have memory requirements approaching the maximum for any algorithm. The total memory required by the pyramid is

$$= \sum_{l=0}^{\log N} 4^{\log N - l} \cdot C_m \cdot 2^l$$

$$= 0(N^2) \text{ words.}$$

2) *Computation Time:* The total computation time for an $N \times N$ image is the sum of the numbers of arithmetic and logical operations at all $\log N + 1$ levels of the pyramid and the sum of the transmission times between all $\log N$ pairs of adjacent levels.

Interior-based algorithms perform a fixed number, say C, of arithmetic operations at each PE in the pyramid and transmit a fixed number of data words per PE. Hence the total number of parallel arithmetic operations and parallel data transfers executed is proportional to the number of levels: $0(\log N)$. Border-based algorithms that count local properties perform a fixed number of arithmetic operations at each PE plus a Boolean operation on the edges requiring $0(n)$ time for an $n \times n$ window. Thus, the total number of arithmetic/logical operations executed is $0(n)$. The connected component counting algorithm has procedures that scan linked lists of length $0(n)$ for an $n \times n$ window. As has been pointed out in Section III-B, for each of the procedures the total number of run accesses and logical operations required is proportional to the number of border runs. Scanning a linked list of length $0(n)$ and relocating pointers in the list requires $0(n)$ arithmetic and logical operations. The time required by border-based algorithms to transmit data for an $n \times n$ window is $0(n)$. Thus, the sum of the computation and data transmission times for border-based algorithms for an $n \times n$ window is $C_t n$, where the constant C_t depends on the algorithm but not on the window size. Thus, the total number of operations for an $N \times N$ image is

$$= \sum_{l=0}^{\log N} C_t 2^l = 0(N).$$

IV. Interleaved Pyramids

The pyramid architecture described in Section II is a natural candidate for pipelining [4] since at any given time an image is processed at a single level. Progressively higher levels define successive segments in the pipeline. For interior-based algorithms, the PE's at every level execute the same number of arithmetic operations, requiring the same amount of time. Pipelining, therefore, results in simultaneous utilization of all the processors in the pyramid. For the border-based algorithms, on the other hand, each PE corresponding to an $n \times n$ window must process a volume $0(n)$ of data. The actual processing time is data dependent and, in general, is proportional to n. The PE at the root of the pyramid has the largest volume of data, and hence takes the longest time to process its window. The delay between the inputs (outputs) of two consecutive images is equal to the time taken by the PE at the root of the pyramid, and is $0(N)$. Pipelining therefore results in high utilization of the PE at the root and progressively lower utilization of PE's at lower levels, the utilization halving with each level down.

A simple method to better utilize the nonapex PE's is to share them among several pyramids. This approach characterizes the second hierarchy, called interleaved pyramids, that achieves the high utilization of the apex processor at all levels. To describe interleaved pyramids, first let $C_t n$ denote, as before, the time required to execute a border-based algorithm on an $n \times n$ window, where C_t is a constant that depends on the algorithm and can be interpreted as the worst case execution time. A PE cannot process windows at intervals less than $C_t n$. This is to synchronize data transfer between levels. In case a PE completes its execution in time less than $C_t n$, it idles for the remaining time. The lower the level, the lower the processor utilization, and hence larger the number of pyramids that can share that level.

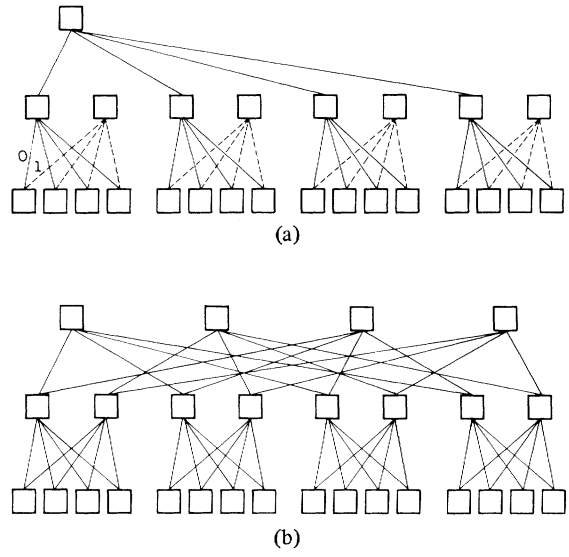


Fig. 4. (a) The basic pyramid (solid lines) with additional nodes (dotted lines) at level 1. (b) Interleaved pyramid with three levels.

Fig. 4(a) shows the basic pyramid drawn in heavy lines. Since a PE at level l has half the workload of its parent at level $l + 1$, and hence is half as busy, it can be time-shared by two parents. Consider the leaf nodes. Let us insert additional nodes at level 1 to double their number. Each leaf now sends data to two different parents alternately [Fig. 4(a)]. Let us say that a leaf initially inputs data to its left parent (position 0), then switches to its right parent (position 1), and then back again. If the leaf nodes transmit data to their parents every C_t time units, then the left and right parents receive data at times 0 and C_t (modulo $2C_t$), respectively. The two sets of parents process different data, received at level 0 at different times. The nodes at level 1 are in turn shared by two parents at level 1 [Fig. 4(b)]. Thus, level 2 has four sets of nodes, each set synchronously transmitting and receiving data every $4C_t$ time units. The different sets receive data in a round robin fashion. The order in which this happens is given by the integers represented by the sequences of position bits along the paths from different sets leading to level 0, level 0 positions representing the least significant bits. Fig 4(b) shows interleaved pyramids containing three levels. In general, an $(L + 1)$-level interleaved pyramid has 2^{2L} input PE's and 2^L output PE's. Nodes at any level l belong to 2^l different sets. The sets time-share the levels below, each set transmitting and receiving data every $2^l C_t$ time units. For an $N \times N$ image the root level has $2^{\log N}$ $(=N)$ nodes each of which outputs its data every NC_t time units, the outputs being interleaved in time and appearing at times $0, C_t, \cdots, (N-1)C_t$. The N nodes represent the roots of N interleaved pyramids, that output results at the same rate at which images are fed in.

The image window to PE mapping used by the interleaved pyramids may be obtained by considering the PE's at a given level in an interleaved pyramid and the nodes at the same level in the complete quadtree representations of the input images. The number of PE's at level l of an interleaved pyramid is $2^{2\log N - l}$. The number of nodes at level l in a single complete quadtree is $4^{\log N - l}$. Thus, nodes at level l in $(2^{2\log N - l})/(4^{\log N - l}) = 2^l$ successive trees maps onto different PE's at level l in the interleaved pyramid. Alternatively, a node in a given position at level l in every $(2^l + 1)$st tree maps onto the same PE at level l. In contrast, the basic pyramid architecture employs a one-to-one mapping between image windows (quadtree nodes) and PE's. Each node in the tree maps onto the PE in the corresponding position in the pyramid.

Since the same computations on tree nodes are performed both by the basic and the interleaved pyramids and only the PE assignment to the nodes changes, the algorithms for the basic pyramids are also valid for the interleaved pyramids.

V. Pyramid Trees

Despite its high throughput rate, the interleaved pyramid presents severe implementation problems, chiefly due to its size. (The same is also true of the basic pyramid.) For example, to process a 512 X 512 image, an interleaved pyramid requires about a half million PE's at ten different levels. Moreover, for maximum speed, the entire image must be fed in parallel. This problem is alleviated by the third hierarchy, called pyramid tree, which provides a way of reducing the required number of PE's and the input memory bandwidth at the cost of a decrease in the throughput rate.

Since the number of PE's in interleaved pyramids grows very fast with the number of levels, shallow interleaved pyramids are desirable. However, interleaved pyramids with fewer than $\log N + 1$ levels can only process images smaller than $N \times N$. Consider interleaved pyramids with k levels. Let $N = 2^L$ and $L = pk$, where p and k are positive integers. The image array of $2^L \times 2^L$ pixels may be represented as a $2^k \times 2^k$ array of $2^{k(p-1)} \times 2^{k(p-1)}$ windows of pixels. If the results of computation and border-related information pertaining to these windows are available, they may be processed by a k-level interleaved pyramid, each of whose bottom level PE's receive data corresponding to four windows. Since a $2^{k(p-1)} \times 2^{k(p-1)}$ window may still be too large to be processed by a single k-level interleaved pyramid, it can be further divided into a $2^k \times 2^k$ array of $2^{k(p-2)} \times 2^{k(p-2)}$ windows of pixels. If such a recursive division is continued for $(p-1)$ steps, $2^k \times 2^k$ arrays of $2^{k[p-(p-1)]} \times 2^{k[p-(p-1)]}$, i.e., $2^k \times 2^k$, windows of pixels are obtained. The final set of $2^k \times 2^k$ windows requires a $(k+1)$-level interleaved pyramid. The required interconnection of interleaved pyramids is a tree (Fig. 5). For a $2^L \times 2^L$ image, where $L = pk$, the tree has p levels. The leaf nodes are $(k+1)$-level interleaved pyramids and accept pixel inputs. Such a two-step hierarchy (tree of interleaved pyramids) of PE's, called pyramid tree, defines the third organization described in this paper. The branching factor of the tree will be discussed below.

Let a PE in the pyramid tree have the index (l_t, l_p), meaning that the PE is located at level l_p of an interleaved pyramid that is itself located at level l_t of the tree. Consider a nonleaf interleaved pyramid at level l_t, $2 \leq l_t \leq p-1$. It has $2^{2(k-1)}$ PE's at level 0 [index $(l_t, 0)$] and 2^{k-1} PE's at level $k-1$ [index $(l_t, k-1)$]. Each PE with index $(l_t, 0)$, $l_t \geq 2$, receives data for four windows, each of size $2^{k(l_t-1)} \times 2^{k(l_t-1)}$, processed by PE's indexed $(l_t - 1, k-1)$. Thus, data for 2^{2k} windows are received from interleaved pyramids each of which has 2^{k-1} output PE's. This requires $(2^{2k})/(2^{k-1}) = 2^{k+1}$ children interleaved pyramids at level $l_t - 1$. However, as will be seen later, for synchronization purposes we will allow only 2^k children (Fig. 5). Thus each child PE indexed $(l_t - 1, k-1)$ sends data for two windows in the $2^k \times 2^k$ array of windows.

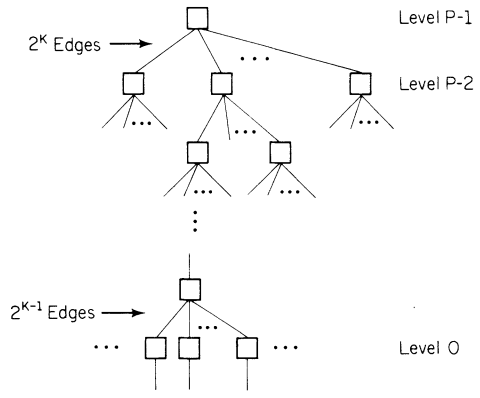
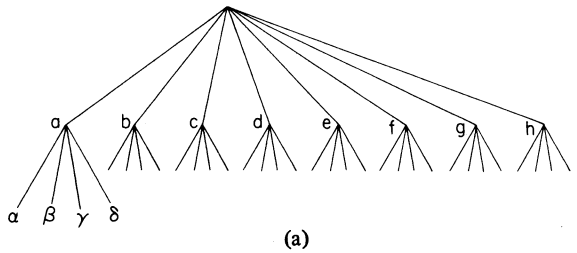


Fig. 5. A pyramid tree.

Now consider the pyramids at level $l_t = 1$. Each such interleaved pyramid must receive data from output PE's of $(k+1)$-level interleaved pyramids at level $l_t = 0$. Since level 0 pyramids have 2^k output PE's each, where each output PE supplies data for two windows serially, the number of children interleaved pyramids per interleaved pyramid at level $l_t = 1$ is $(2^{2k})/(2^k \cdot 2) = 2^{k-1}$ (Fig. 5). PE's indexed $(l_t, 0)$, $l_t \geq 1$, receive four (window) inputs each, whereas PE's with index (0, 0) receive single pixel inputs. Fig. 6(a) shows a schematic of a pyramid tree for a 512 X 512 image with $k = p = 3$. The root node and its $2^k = 8$ children nodes constitute the top $p - 1 = 2$ levels of the pyramid tree. Thus, each of these nodes represents a three-level interleaved pyramid. Each of the level-1 nodes has $2^{k-1} = 4$ children nodes, each representing a $k + 1 = 4$ level interleaved pyramid.

Fig. 6(b) shows a 512 X 512 pixel image divided into an 8 X 8 ($2^k \times 2^k$) array of 64 X 64 windows. These windows are processed in two batches (for synchronization) by the level-1 pyramids to provide inputs to the root node. The division of the windows into two batches and the order in which the windows within each batch are processed is not unique; one way is shown in Fig. 6(b). In this example an 8 X 8 array is divided into its northern and southern halves. The windows in the northern half are processed prior to those in the southern half. In either half, each of the $2^k = 8$ level-1 interleaved pyramids processes all windows from a single column in the order shown by the integer labels.

Fig. 6(c) represents one of the 64 X 64 windows divided into an 8 X 8 array of subwindows to be processed by $2^{k-1} = 4$ of the bottom level interleaved pyramids that have a common parent. As in Fig. 6(b), the array is divided into its northern



(a)

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

(b)

α	β	γ	δ	α	β	γ	δ
1	1	1	1	2	2	2	2
3	3	3	3	4	4	4	4
5	5	5	5	6	6	6	6
7	7	7	7	8	8	8	8
1	1	1	1	2	2	2	2
3	3	3	3	4	4	4	4
5	5	5	5	6	6	6	6
7	7	7	7	8	8	8	8

(c)

Fig. 6. (a) A schematic for a pyramid tree with $k = p = 3$. Each node represents an interleaved pyramid. (b) An 8 X 8 array of 64 X 64 windows of a 512 X 512 image to be processed by the interleaved pyramids at level 1. (c) An 8 X 8 array of 8 X 8 windows to be processed by a set of four level-0 interleaved pyramids.

and southern halves. In either half, each of the four interleaved pyramids processes all the windows from two columns with the same column-header and in the order shown by the integer labels.

It will now be seen that such a serial-parallel mode of dataflow does not render any node idle due to lack of data. This is because the time required by a node to process current data is also the time required to receive the data for the following computation. In the following, l_t is denoted by l for short. An interleaved pyramid represented by a node at level l, $1 \leqslant l \leqslant p - 1$, of the tree has inputs that correspond to windows of size $2^{kl} \times 2^{kl}$. The processing time in a PE at the base of that pyramid is $C_t \cdot 2^{kl+1}$, and therefore the output latency of that pyramid is $C_t \cdot 2^{kl+1}$, provided the inputs arrive at the same rate (Section III-B). This node's 2^{2k} inputs are divided into 2^k groups each of size 2^k with inputs within a group arriving serially since they are the interleaved outputs (each output being used twice) of a single interleaved pyramid one level below. Thus, all the inputs arrive in the time required to produce 2^k interleaved outputs serially in an interleaved pyramid at level $l - 1$. Using an inductive argument on l for $l \geqslant 1$, the output latency of such a pyramid at level $l - 1$ is $C_t \cdot 2^{k(l-1)+1}$, and hence the total time to produce all 2^k outputs within a group is $C_t \cdot 2^{kl+1}$ as is required.

The mapping between image windows and PE's in a pyramid tree may be obtained in two steps. First, the mapping between image windows and interleaved pyramids in the tree is given by the recursive division of the image into $2^k \times 2^k$ subarray of windows, as described earlier in this section. Next, within an interleaved pyramid, the image window to PE mapping is the same as described in Section V. Together, the two steps determine the general image window to PE mapping in pyramid trees. Algorithms designed for the basic pyramids may also be used by PE's in pyramid trees.

VI. Performance Measures

This section evaluates 1) the number P of PE's and 2) the total memory M required by the basic (pipelined) pyramid, the interleaved pyramid, and the pyramid tree. The performance of the architectures is compared with respect to 3) the input bandwidth B, 4) the output latency L (reciprocal of the throughout), and 5) the image turnaround time T. The input bandwidth measures the degree of parallelism of dataflow at the input. As seen from the discussion in Sections IV and V, the interleaved pyramids and pyramid trees offer reduced output latencies partly at the expense of increased PE and memory requirements. To compare the three architectures with respect to the effective PE and memory utilization, the above measures 1) and 2) will be normalized with respect to 4) to obtain two additional measures. The first of these measures P_T, is the number of PE's used per unit output latency. This measure reflects overall processor utilization—the smaller its value, the better the utilization. A deficiency of this measure is that it assigns the same weight to PE's at all levels and does not take into account their different memory sizes. To remedy this, the total memory M_T required per unit output latency is computed. Taking their weighted sum in some technology dependent proportion, P_T and M_T together measure the overall efficiency of a design.

A. Basic Pyramid

1) Number of Processors:

$$P = \sum_{l=0}^{\log N} 4^l \approx \frac{4N^2}{3}.$$

2) Total Memory:

$$M = \sum_{l=0}^{\log N} 4^{\log N - l} \cdot C_m \cdot 2^l$$

$$= C_m N(2N - 1).$$

3) Input Bandwidth: The input bandwidth is the number of parallel pixel inputs to the PE's at the bottom level of the pyramid. Since there are as many PE's at the bottom level as the number of pixels, and all pixels are loaded in parallel, B is given by

$$B = 4^{\log N} = N^2.$$

4) Output Latency: The pipelining results in an interval of C time units [see Section III-C-2] between successive outputs of the interior-based algorithms. For the border-based algorithms, the PE at the root of the pyramid takes the longest time to process its window. Therefore the delay between two consecutive input (output) images is the time taken by the PE at

the root of the pyramid. Therefore

$L = C$, for interior-based algorithms

$= C_t N$, for border-based algorithms.

5) Image Turnaround Time: The image turnaround time is the time required to process an entire image. This is simply the sum of the processing times required at all the levels of the pyramid. For the interior-based algorithms,

$T = C(\log N + 1)$.

For the border-based algorithms,

$$T = \sum_{l=0}^{\log N} C_t \cdot 2^l$$

$$= C_t \cdot (2N - 1).$$

6) Number of Processors Per Unit Output Latency:

$$P_T = C \cdot \left(\frac{4N^2}{3}\right) = \frac{4}{3} CN^2, \quad \text{for interior-based algorithms}$$

$$= C_t N \cdot (4N^2/3) = \frac{4}{3} C_t N^3, \quad \text{for border based algorithms.}$$

7) Total Memory Per Unit Output Latency:

$M_T = C \cdot C_m N(2N - 1)$, for interior-based algorithms

$= C_m N(2N - 1) C_t N = C_t C_m N^2 (2N - 1)$,

for border-based algorithms.

B. Interleaved Pyramid

The interleaved pyramid has a performance similar to the basic pipelined pyramid, for processing interior-based algorithms. For border-based algorithms, it provides the same kind of improvement in performance over the basic pipeline pyramid, as pipelining provides over the basic pyramid for processing interior-based algorithms—both increase utilization of lower level PE's and the throughput rate. However, the interleaved pyramid uses many more PE's than the basic pyramid, although the computation time and memory size required at a single PE corresponding to an $n \times n$ window remain $C_t n$ and $C_m n$, respectively, as in the basic pyramid. The performance measures described earlier are computed below for the interleaved pyramids executing the border-based algorithms.

1) Number of Processors:

$$P = \sum_{l=0}^{\log N} 2^{2 \log N - l}$$

$$= N(2N - 1).$$

2) Total Memory:

$$M = \sum_{l=0}^{\log N} \text{(number of nodes at level } l\text{)} \cdot \text{(memory required per node at level } l\text{)}$$

$$= \sum_{l=0}^{\log N} 2^{2 \log N - l} \cdot C_m 2^l$$

$$= C_m N^2 \log 2N.$$

3) Input Bandwidth:

$B = N^2$ (same as for the basic pyramid).

4) Output Latency:

$L = C_t$.

5) Image Turnaround Time:

$T = C_t(2N - 1)$ (same as for the basic pyramid).

6) Number of Processors Per Unit Output Latency:

$P_T = C_t \cdot (2N^2 - N)$.

7) Total Memory Per Unit Output Latency:

$M_T = C_t C_m \cdot N^2 \log 2N$.

Note that the output latency is equal to the processing time of a PE at the base of the pyramid. The expressions for P_T and M_T indicate that the interleaved pyramid achieves a factor of N improvement in processor utilization and a factor of $N/\log N$ improvement in memory utilization over the pipelined basic pyramid. This is achieved by approximately a $\frac{3}{2}$ increase in the number of processors and a $\log N$ increase in the amount of memory.

C. Pyramid Tree

The performance of the pyramid tree is a function of the parameters p and k. The derivations of the performance measures are more involved than for the previous two cases. Therefore, only the results are listed below; the detailed derivations are provided in the Appendix. Once again, the performance measures obtained are for the more complex case of the border-based algorithms.

1) Number of Processors: Let $P(p, k)$ denote the total number of PE's in the pyramid tree required to process an $N \times N$ image ($N = 2^{pk}$). Then,

$$P(p,k) = \begin{cases} N \cdot 2^k - 2^{k-1} & \text{for } p \geq 2 \\ N(2N - 1) & \text{for } p = 1. \end{cases}$$

The following table gives values of $P(p, k)$ for small values of p and k. It indicates the large reduction in hardware as compared to the other two architectures.

k \ p	1	2	3	4
2	28	62	254	1022
3	120	508	4092	32 764
4	496	4088	65 528	1.049×10^6

2) Total Memory: The memory $M(p, k)$ required by the pyramid tree can be expressed as

$M(l, k) = C_m \cdot 2^{2k}(k + 1)$

and

$$M(p,k) = \frac{C_m}{2} \cdot (k + 1) \cdot p \cdot 2^{k(p+1)} \quad \text{for } p > 1.$$

TABLE I

	# of PE's	Total memory size	Output latency	InPut Bandwidth	Turnaround time	PE_T	M_T
Basic Pyramid	$\approx \frac{4N^2}{3}$	$O(N^2)$	$O(N)$	$O(N^2)$	$O(N)$	$O(N^3)$	$O(N^3)$
Interleaved Pyramid	$2N^2-N$	$O(N^2 \log N)$	$O(1)$	$O(N^2)$	$O(N)$	$O(N^2)$	$O(N^2 \log N)$
Pyramid Tree ($p > 1$)	$\approx N^{1+1/p}$	$O(N^{1+1/p} \log N)$	$O(N^{1-1/p})$	$O(N^{1+1/p})$	$O(N+N^{1-1/p})$	$O(N^2)$	$O(N^2 \log N)$

32

Normalized (relative to C_m) values of $M(p, k)$ for several small values of p and k are given as follows.

k \ p	1	2	3	4
2	48	176	104	5376
3	256	1920	22 528	237 568
4	1280	19 456	448×2^{10}	37×2^{18}

3) Input Bandwidth:

$$B = \frac{N}{2} \cdot 2^k.$$

4) Output Latency:

$$L = 2C_t \cdot N/2^k.$$

5) Image Turnaround Time:

$$T < 2C_t \cdot (N + N^{1-1/p}).$$

6) Number of Processors Per Unit Output Latency:

$$P_T = C_t \cdot N(2N - 1).$$

7) Total Memory Per Unit Output Latency:

$$M_T < C_m C_t p(k+1)N^2$$
$$< 2C_m C_t N^2 \log N.$$

These expressions indicate that the asymptotic performance of the pyramid tree is the same or better compared to the basic pipelined pyramid, although at a substantial reduction in the number of PE's, memory, and input bandwidth.

VII. Conclusions

This paper describes three multiprocessor pyramid architectures for bottom-up image analysis: pyramids, interleaved pyramids, and pyramid trees. Bottom-up analysis is made feasible by transmitting up the pyramid levels quadrant borders and border-related information that captures quadrant interaction of interest for a given computation. A small general-purpose processor is used as the processing element (PE).

Since output is available only at the apex PE's, the only time-efficient computations are those whose results are expressible in a small number of bits, to keep the data output operation from dominating the overall processing time. The algorithms described in the paper compute $O(\log N)$ bit results on an $N \times N$ image. However, other computations can be performed if the time required for outputting the results from apex PE('s) is acceptable (e.g., when the basic pyramid is not pipelined, or when slow loading of input image makes the outputting time at the apex irrelevant). Certain transformations on point patterns constitute one class of such computations. For instance, the Voronoi tessellation of the plane defined by a set of points (nuclei) is the partition of the plane into cells in which each cell contains exactly one nucleus, and all points within the cell are closer to this nucleus then to any other. Similarly, the minimal spanning tree defined by a set of points is a tree that spans all points while minimizing the total cost, say the sum of edge lengths. Such transformations on point patterns are useful in a wide variety of applications [27], and involve specifying $O(N \log N)$ bits for a pattern consisting of N points [$O(\log N)$ bits for each of the line segments defining the desired graph]. The architectures described in this paper also offer the possibility of using four-way divide-and-conquer based algorithms, instead of the often used two-way divide-and-conquer paradigm, thus reducing the computation time requirements of the above transformations from $O(N \log_2 N)$ to $O(N \log_4 N)$.

Interleaved pyramids result in high utilization of PE's at all levels, unlike the basic pyramids where PE's at lower levels are increasingly idle. The pyramid tree offers a solution to the problems of large input bandwidth and large number of PE's required for a purely interleaved pyramid. The designer is provided with parameters that can be manipulated to achieve suitable tradeoffs between the hardware cost and processing time.

Table I summarizes the performance measures 1)-7) for all three architectures described in this paper. One note of caution must, however, be added. The implementation of pipelining and interleaving assumes that images can be loaded in parallel and at as fast a rate as desired. This is not possible

if, for example, one does not have parallel access to the input image pixels (e.g., outputs of individual sensors). In such cases the input image loading time may dominate the processing time, and hence may make fast processing capabilities superfluous. Parallel optical transmission of the sensor information using fiber optics is one solution to designing the focal plane architectures [7] which will make parallel loading of images possible.

APPENDIX

The performance measures given in Section VI-C are derived below, for the pyramid executing the border-based algorithms.

1) Number of Processors: Let $P(p, k)$ be the total number of PE's in the pyramid tree required to process an $N \times N$ image ($N = 2^{pk}$). Recall that each of the level-1 nodes has 2^{k-1} children nodes, and all higher level nodes have 2^k children each. The number of k-level interleaved pyramids at level l, $1 \leqslant l \leqslant p-1$, is thus $2^{k(p-1-l)}$. The number of level-1 nodes is $2^{k(p-2)}$, and hence, the number of leaf nodes (each of which is a $k+1$-level interleaved pyramid) is $2^{k(p-2)} \cdot 2^{k-1}$. If $H(k)$ denotes the number of PE's in a k-level interleaved pyramid, then

$$P(p,k) = \begin{cases} H(k+1) \cdot 2^{k(p-2)+k-1} + H(k) \cdot \sum_{l=1}^{p-1} 2^{k(p-1-l)} & \\ \qquad \text{for } p \geqslant 2 & \\ H(k+1) \quad \text{for } p = 1. \end{cases}$$

Now, from Section III-B-1, $H(k) = 2^{k-1}(2^k - 1)$. Substituting for $H(k)$ and $H(k+1)$ in the expression for $P(p, k)$ gives

$$P(p,k) = \begin{cases} N \cdot 2^k - 2^{k-1} & \text{for } p \geqslant 2 \\ N(2N-1) & \text{for } p = 1. \end{cases}$$

2) Total Memory: Let $M(p, k)$ be the total memory required by the pyramid tree. To compute $M(p, k)$, we will compute the memory requirements of a nonleaf node and a leaf node separately. A nonleaf node at level l, $l \geqslant 1$, of the tree is a k-level interleaved pyramid which, according to the result of the previous section, requires a memory of $C_m \cdot 2^{2(k-1)} \cdot k \cdot 2^{lk+1}$. Here, the factor 2^{lk+1} occurs because of the fact that the inputs are windows of size $2^{lk} \times 2^{lk}$ rather than single pixels. However, unlike the interleaved pyramid of Section IV, the interleaved pyramid at level l of the tree does not receive all its inputs in parallel and therefore must use extra memory in the PE's at its base to store serially arriving inputs of size 2^{lk}. There are $2^{2(k-1)}$ PE's at the base and each PE needs an extra memory of $C_m 2^{lk+1}$. The total memory requirement of a nonleaf node at level l is therefore

$$= C_m \cdot 2^{2(k-1)} \cdot k \cdot 2^{lk+1} + C_m \cdot 2^{2(k-1)} 2^{lk}$$

$$= \frac{C_m}{2} \cdot 2^{k(l+2)}(k+1).$$

There are $2^{k \cdot (p-1-l)}$ such nodes at a level l, $1 \leqslant l \leqslant p-1$. Among the $2^{k(p-2)+(k-1)}$ leaf nodes, each is a $(k+1)$ level interleaved pyramid requiring a memory of $C_m \cdot 2^{2k} \cdot (k+1)$. Thus, the memory required by the pyramid tree to process an $N \times N$ image, $M(p, k)$, is obtained by adding the memory requirements of the nonleaf and leaf nodes.

For the case $p > 1$

$$M(p,k) = \frac{C_m}{2}(k+1) \sum_{l=1}^{p-1} 2^{k(p-1-l)} \cdot 2^{k(l+2)} + C_m \cdot 2^{k(p-2)+(k-1)} \cdot 2^{2k}(k+1)$$

$$= C_m 2^{k(p+1)} \left(\frac{(k+1)}{2}(p-1) + \frac{(k+1)}{2} \right)$$

$$= \frac{C_m}{2} \cdot (k+1) \cdot p \cdot 2^{k(p+1)}.$$

For the case $p = 1$

$$M(l,k) = C_m 2^{2k}(k+1).$$

3) Input Bandwidth: The input bandwidth is the number of parallel pixel inputs. Since the interleaved pyramids corresponding to the leaf nodes of the tree accept single pixel inputs, the total number of PE's at the base of the $2^{k(p-2)+k-1}$ level-0 pyramids in the tree must equal the bandwidth. Thus,

B = (number of interleaved pyramids at level 0 of the pyramid tree) $\cdot 2^{2k}$

$$= 2^{k(p-2)+k-1} \cdot 2^{2k}$$

$$= 2^{kp+k-1} = \frac{N}{2} \cdot 2^k.$$

4) Output Latency: To compute the output latency of the pyramid tree, observe that the outputs of pyramids at any given level of the tree occur in synchronism. Therefore, the entire tree may be viewed as a segmented pipeline with the delay at a given segment equal to the output latency of an interleaved pyramid at that level. The maximum such delay occurs at the interleaved pyramid at the root of the tree.

The PE's comprising the base of the interleaved pyramid at the root of the tree process borders of size $2^{k(p-1)+1} \times 2^{k(p-1)+1}$, and hence require processing time $C_t \cdot 2^{k(p-1)+1}$. The interleaved pyramid is thus able to accept new inputs at a maximum rate of $C_t \cdot 2^{k(p-1)+1}$. Such an input rate is actually achieved because the output latency of an interleaved pyramid one level down (whose 2^{k-1} outputs are fed serially to the pyramid at the root) is $C_t \cdot 2^{k(p-2)+1}$; hence, 2^k output values (with each output PE used twice) are available at the rate of $C_t \cdot 2^{k(p-1)+1}$, the maximum rate at which inputs are accepted by the interleaved pyramid at the root. From Sections IV and VI-B, it follows that the output latency of the interleaved pyramid at the root is $C_t \cdot 2^{k(p-1)+1}$. Therefore,

L = output latency of the interleaved pyramid at the root of the tree

$$= C_t \cdot 2^{k(p-1)+1}$$

$$= 2C_t \cdot N/2^k.$$

5) Turnaround Time: The turnaround time is the total time required to process an image. Let us first compute the time taken by a nonleaf node at level l, $1 \leqslant l \leqslant p-1$, to process a set of 2^{2k} input windows each of size $2^{kl} \times 2^{kl}$, which arrive

in parallel in batches of size 2^k. From the discussion at the beginning of this section, the output latency of nodes one level down, i.e., at level $l-1$, is $C_t \cdot 2^{k(l-1)+1}$. Hence, 2^k batches of inputs to nodes at level l arrive serially in time $C_t \cdot 2^{k(l-1)+1} \cdot 2^k$. These inputs are then processed by the k-level interleaved pyramid with the PE's at level $j, 0 \leqslant j \leqslant k-1$, taking time $C_t \cdot 2^{kl+1+j}$. The total time required by the node is thus the sum of the time required to collect all the inputs and the time required to process these inputs through the k levels. Thus, the total time is

$$= C_t \cdot 2^{kl+1} + C_t \cdot \sum_{j=0}^{k-1} 2^{kl+1+j}$$

$$= C_t \cdot 2^{k+1} \cdot 2^{kl}.$$

The single pixel inputs to a leaf node in the pyramid tree arrive in parallel and are processed by the $k+1$ levels of the interleaved pyramid in total time $C_t \cdot (2^{k+1}-1)$. Since the input bandwidth of the pyramid tree is $N/2 \cdot 2^k$, the processing time for a single set of $N/2 \cdot 2^k$ pixel inputs is

$$= C_t \cdot (2^{k+1}-1) + C_t \cdot 2^{k+1} \cdot \sum_{l=1}^{p-1} 2^{kl}$$

$$< 2C_t \cdot N.$$

There are $2N^2/N2^k$ sets of pixel inputs per image occurring at intervals of time C_t. The turnaround time for the entire image is therefore given by

$$T \approx C_t \cdot 2N/2^k + 2C_t \cdot N$$

$$= 2C_t \cdot (N + N^{1-1/p}).$$

6) Number of Processors Per Unit Output Latency: P_T is obtained as the product of 1) and 4). Thus,

$$P_T = (2N-1)2^{k-1} \cdot 2C_t \cdot N/2^k$$

$$= C_t \cdot N(2N-1) = 0(N^2).$$

7) Total Memory Per Unit Output Latency: M_T is obtained as the product of 2) and 4). Thus,

$$M_T < \frac{C_m}{2} \cdot N \cdot 2^k \cdot p \cdot (k+1) \cdot 2C_t \cdot N/2^k$$

$$< C_m \cdot C_t \cdot p(k+1) \cdot N^2$$

$$= 0(N^2 \log N).$$

References

[1] L. Uhr, "Layered 'recognition cone' networks that preprocess, classify, and describe," *IEEE Trans. Comput.*, vol. C-21, pp. 758–768, 1972.

[2] A. R. Hanson and E. M. Riseman, Eds., *Computer Vision Systems*. New York: Academic, 1978.

[3] L. Uhr, M. Thompson, and J. Lackey, "A 2-layered SIMD/MIMD parallel pyramidal array/NET," in *Proc. IEEE Workshop Comput. Architecture Pattern Anal. Image Database Management*, Hot Springs, AK, 1981, pp. 209–216.

[4] L. Uhr, "Converging pyramids of arrays," in *Proc. IEEE Workshop Comput. Architecture Pattern Anal. Image Database Management*, Hot Springs, AK, 1981, pp. 31–34.

[5] C. R. Dyer, "Cellular pyramids for image analysis," Ph.D. dissertation, Dep. Comput. Sci., Univ. Maryland, College Park, MD, 1978.

[6] A. Nakamura and C. R. Dyer, "Bottom-up cellular pyramids for image analysis," in *Proc. 4th Int. Joint Conf. Pattern Recognition*, Kyoto, Japan, Nov. 7–10, 1978, pp. 474–496.

[7] C. R. Dyer, "A VLSI pyramid machine for hierarchical parallel image processing," in *Proc. Pattern Recognition Image Processing Conf.*, Dallas, TX, Aug. 3–5, 1981, pp. 381–386.

[8] A. Rosenfeld, *Picture Languages*. New York: Academic, 1979.

[9] —, "Quadtrees and pyramids for pattern recognition and image processing," in *Proc. 5th Int. Conf. Pattern Recognition*, 1980, pp. 802–811.

[10] H. Samet and A. Rosenfeld, "Quadtree representations of binary images," in *Proc. 5th Int. Conf. Pattern Recognition*, Miami Beach, FL, Dec. 1–4, 1980, pp. 815–818.

[11] N. Ahuja, "Approaches to recursive image decomposition," in *Proc. Pattern Recognition Image Processing Conf.*, Dallas, TX, Aug. 3–5, 1981, pp. 75–80.

[12] M. D. Kelly, "Edge detection in pictures by computer using planning," *Machine Intell.*, vol. 6, pp. 379–409, 1971.

[13] M. D. Levine, "A knowledge based computer vision system," in *Computer Vision Systems*, A. Hanson and E. Riseman, Eds. New York: Academic, 1978, pp. 335–352.

[14] K. Price and R. Reddy, "Change detection and analysis in multispectral images," in *Proc. 5th Int. Joint Conf. Artificial Intell.*, Cambridge, MA, 1977, pp. 619–625.

[15] S. L. Tanimoto, "Pictoral feature distortion in a pyramid," *Comput. Graphics Image Processing*, vol. 5, pp. 333–352, Sept. 1976.

[16] S. L. Tanimoto and T. Pavlidis, "A hierarchical data structure for picture processing," *Comput. Graphics Image Processing*, pp. 104–119, June 1975.

[17] P. J. Burt, T. Hong, and A. Rosenfeld, "Segmentation and estimation of image region properties through cooperative hierarchical computation," Univ. Maryland, College Park, MD, Comput. Sci. Tech. Rep. TR-927, Aug. 1980.

[18] M. Schneier, "Two hierarchical linear feature representations: Edge pyramids and edge quadtrees," Univ. Maryland, College Park, MD, Comput. Sci. Tech. Rep. TR-961, Oct. 1980.

[19] T. Dubitzki, A. Wu, and A. Rosenfeld, "Parallel region property computation by active quadtree networks," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-3, pp. 626–633, Nov. 1981.

[20] K. Batcher, "Bit-serial parallel processing systems," *IEEE Trans. Comput.*, vol. C-31, pp. 377–384, May 1982.

[21] M. J. B. Duff, "CLIP 4: A large scale integrated circuit array parallel processor," in *Proc. 3rd Int. Conf. Pattern Recognition*, 1976, pp. 728–733.

[22] S. F. Reddaway, "DAP–A distributed processor array," in *Proc. 1st Annu. Symp. Comput. Architecture*, Dec. 1983, pp. 61–65.

[23] C. Rieger, "ZMOB: Doing it in parallel," in *Proc. IEEE Workshop Comput. Architecture Pattern Anal. Image Database Management*, Nov. 1981, pp. 119–124.

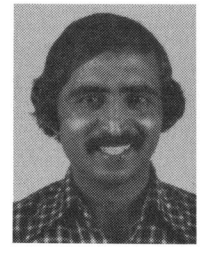
[24] H. J. Siegel, L. J. Siegel, F. C. Kemmerer, P. T. Mueller Jr., H. E. Smalley Jr., and S. D. Smith, "PASM: A partitionable SIMD/MIMD system for image processing and pattern recognition," *IEEE Trans. Comput.*, vol. C-30, pp. 934–947, Dec. 1981.

[25] A. Rosenfeld and A. C. Kak, *Digital Picture Processing*. New York: Academic, 1976.

[26] C. R. Dyer, "Computing the Euler number of an image from its quadtree," Univ. Maryland, College Park, MD, Comput. Sci. Tech. Rep. TR-769, May 1979.

[27] N. Ahuja, "Dot pattern processing using Voronoi neighborhoods," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-4, pp. 336–343, May 1982.

[28] S. Swamy and N. Ahuja, "A bottom-up connected component counting algorithm without using labels," unpublished manuscript.

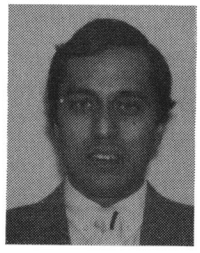


Narendra Ahuja (S'79–M'79) received the B.E. degree with honors in electronics engineering from the Birla Institute of Technology and Science, Pilani, India, in 1972, the M.E. degree with distinction in electrical communication engineering from the Indian Institute of Science, Bangalore, India, in 1974, and the Ph.D. degree in computer science from the University of Maryland, College Park, in 1979.

From 1974 to 1975 he was Scientific Officer in the Department of Electronics, Government

of India, New Delhi. From 1975 to 1979 he was at the Computer Vision Laboratory, University, of Maryland, College Park, as a Graduate Research Assistant (1975–1978), as a Faculty Research Assistant (1978–1979), and as Research Associate (1979). In 1979 he joined the University of Illinois, Urbana-Champaign, where he is currently a Research Assistant Professor in the Coordinated Science Laboratory and an Assistant Professor in the Department of Electrical Engineering. His research interests are in computer vision, artificial intelligence, image processing, architectures for vision, and parallel algorithms.

Dr. Ahuja is listed in *Who's Who in Technology Today* and *Who's Who in the Midwest*, and is a member of the Association for Computing Machinery and the American Association for Artificial Intelligence.



Sowmitri Swamy received the Ph.D. in computer science from Brown University, Providence, RI, in 1978.

He then joined the Department of Electrical Engineering/Coordinated Science Laboratory at the University of Illinois at Urbana-Champaign as an Assistant Professor. In 1982 he joined the Department of Computer Science, Gould Research Center, Rolling Meadows, IL, where he is currently the Principal Investigator for the Distributed Computer System Design project. His current interests are image processing architectures, resource allocation in distributed systems, and fault tolerant embedded systems.

Picture Indexing and Abstraction Techniques for Pictorial Databases

SHI-KUO CHANG, SENIOR MEMBER, IEEE AND SHAO-HUNG LIU

Abstract—**We present an approach for picture indexing and abstraction. Picture indexing facilitates information retrieval from a pictorial database consisting of picture objects and picture relations. To construct picture indexes, abstraction operations to perform picture object clustering and classification are formulated. To substantiate the abstraction operations, we also formalize syntactic abstraction rules and semantic abstraction rules. We then illustrate by examples how to apply these abstraction operations to obtain various picture indexes, and how to construct icons to facilitate accessing of pictorial data.**

Index Terms—**Database abstraction, pictorial database, pictorial information retrieval, picture indexing.**

I. Introduction

ADVANCES in hardware technology have paved the way for sophisticated pictorial information systems, with application areas including knowledge-based image understanding systems, office information systems, and integrated manufacturing information systems. A common requirement of these systems is the need to model and access pictorial data with ease. Researchers working on pictorial information systems have developed the concept of logical pictures, which consist of picture objects and picture relations [8], [29], [35]. The relational database approach has also been extended in developing pictorial database models and pictorial query languages [4]-[6].

Manuscript received June 14, 1982; revised May 17, 1983, and February 29, 1984. This work was supported in part by the National Science Foundation under Grant ECS-8005953, and by the Office of Naval Research under Contract N00014-80-C-0651.

S. Chang is with the Information Systems Laboratory, Illinois Institute of Technology, Chicago, IL 60616.

S.-H. Liu is with the Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA 94720.

This paper presents an approach for picture indexing and abstraction. Picture indexing facilitates pictorial information retrieval from a pictorial database. In traditional database systems, the use of indexing to facilitate database accessing has been well established. Although there were suggestions to use picture icons as picture indexes [32], no theoretical framework has been established for picture indexing. In this paper, we attempt to provide such a conceptual framework for picture abstraction, indexing, and retrieval.

In Section II, picture objects, picture relations, and logical pictures are introduced. In Section III, we discuss picture query. Section IV presents structured picture retrieval using picture trees. Examples are presented to motivate the concept of picture indexing. In Section V, two types of abstraction operations are introduced. Type-1 abstraction performs clustering and indexing, and type-2 abstraction performs classification and cross-indexing. Conceptually, type-1 abstraction performs generalization and integration, and type-2 abstraction performs differentiation. They can be recursively applied to obtain various picture indexes. In Sections VI and VII, we present abstraction rules, which include both syntactic abstraction rules and semantic abstraction rules. In Section VIII, we illustrate by example how to construct icons to facilitate accessing of pictorial data, and discuss applications of the proposed approach for picture indexing and abstraction.

II. The Logical Picture

Researchers in image processing and pattern recognition have traditionally regarded pictures as two-dimensional arrays of pixels (or picture elements), which are called *physical pictures*

0162-8828/84/0700-0475$01.00