A Pipelined Architecture for Parallel Image Relaxation Operations

WEI WANG, JUN GU, STUDENT MEMBER, IEEE, AND THOMAS C. HENDERSON, SENIOR MEMBER, IEEE

Abstract—**Discrete relaxation techniques have proven useful in solving a wide range of problems in digital image processing, computer vision, and robot vision. A conventional hardware design for an 8-object, 8-label Discrete Relaxation Algorithm (DRA) requires three 4K memory blocks and maximum execution time of over an hour, which makes such a DRA hardware implementation infeasible. By reformulating the Discrete Relaxation Algorithm into a parallel computational tree, a pipelined Single Instruction stream Multiple Data stream (SIMD) architecture for a highly concurrent computation of an 8-object, 8-label DRA problem has been developed. We give a second implementation which eliminates the excessive memory requirements and performs the DRA computation in microseconds, at the worst case in milliseconds. The chip is fabricated using a 3-μm NMOS technology by MOSIS. The major design issues are described in this paper.**

I. Introduction

THE DISCRETE Relaxation Algorithm (DRA) is a very general computational technique for a wide range of theoretical and engineering problems. Since its invention many years ago, it has demonstrated powerful and extensive applications in many areas. Some of them are listed below:

1. *Digital Image Processing*: for digital image filtering, particularly in the restoration and identification of moving objects from ambiguous environment;
2. *Artificial Intelligence*: propagating numeric constraints among each object being imaged and performing heuristic search for optimal scene decomposition;
3. *Computer Vision*: dealing with the problems such as graph homomorphism, graph coloring, and image understanding; for line finding, stereopsis, line labeling, and semantics-based region growing, etc.;
4. *Robotics*: for solving its motion and vision problems.

For a review of the numerous applications of relaxation processes see [1]–[6], [8].

Classical relaxation (CR) was introduced by Southwell in 1940 [3] and the symbolic (as opposed to numeric) versions of relaxation (SR) were introduced in the mid-seventies [4]. The version used here is that described by Henderson [5]. The *Discrete Relaxation Algorithm* (*DRA*) is a restriction of the classical relaxation process to systems of Boolean inequalities which take values over the two element set $\{0,1\}$. One of the significant techniques resulting from the introduction of the DRA is that these relaxation algorithms are directly executable in silicon subroutines, thus making many real-time digital image relaxation applications feasible.

While most of the work on solving the image relaxation problem has been for single processor systems, much work has been devoted to develop parallel architectures. Due to the higher order of computational cost, including space complexity, time complexity, and data communication costs, current research in this aspect is blocked and has only appeared in a virtual software simulator format. The project described in this paper is a hardware implementation of this algorithm.

In Section II, we will briefly describe the Discrete Relaxation Algorithm; an example for eliminating the ambiguity in the region coloring problem is given. Then, we will define the DRA hardware implementation problem in Section III. The complexity analyses of the DRA hardware implementation and its parallel reformulation are discussed in Section IV. In Section V, the major design issues for the DRA2 chip are presented. Finally, some comparisons of the DRA2 with a conventional DRA1 chip design are given.

II. Discrete Relaxation Algorithm (DRA)

A. Boolean Formulation of Discrete Relaxation Algorithm

Instead of seeking a real number solution in a numerical relaxation situation [3], the solution to be found in the discrete relaxation case involves the assignment of a set of *labels* at each unknown such that some constraint relation among the labels is satisfied by neighboring unknowns [1], [5]. Whereas the unknowns in numerical relaxation take on real number values, the unknowns in a *labeling problem* take on a Boolean vector value with each element in the vector corresponding to a possible label.

The generalized problem involves a set of unknowns which usually represents a set of objects to be given names, a set of labels which are the possible names for the unknown, and a *compatibility model* containing ordered groups of units which mutually constrain one another and ordered groups of unit-label pairs which are compatible. The compatibility model is sometimes called a *world model*. This model tells us which objects mutually constrain one

Manuscript received January 6, 1987; revised June 4, 1987. This work is supported in part by National Science Fountain Grants MCS-82-21750, DCR-85-06393, and DMC-85-02115; and in part by the University of Utah Research Fellowship.

W. Wang is with the Department of Electrical Engineering, University of Utah, Salt Lake City, UT 84112.

J. Gu and T. C. Henderson are with the Department of Computer Science, University of Utah, Salt Lake City, UT.

IEEE Log Number 8716557.

another at a time and which labelings are permitted or legal for those objects which do constrain one another. The problem is to find a label for each object such that the resulting set of object-label pairs is consistent with the constraints of the world model.

Boolean vector operations are denoted by $'$, $\times$, t, $*$, $+$ and $\cdot$ which represent complementation, vector multiplication, transpose, Boolean "and," Boolean "or," and Boolean vector dot product, respectively. Let

1. $U=\{u_1,\cdots,u_n\}$ be the set of unknowns,
2. $\Lambda=\{\lambda_1,\cdots,\lambda_m\}$ be the set of possible labels,
3. $\Lambda_i=(l_1,\cdots,l_m)^t$ be the column vector describing the set of labels (i.e., zero or one) possible for u_i, where $l_j=1$ if λ_j is compatible with u_i; 0 otherwise.
4. C be an m by m compatibility matrix for label pairs, where $C(i,j)=1$ if λ_i is compatible with λ_j; 0 otherwise.
5. $\Lambda_{ij}=(\Lambda_i\times\Lambda_j')*((\mathrm{Nei}(i,j)'E)+C)$ be an m by m compatibility matrix for u_i and u_j, where E is the m by m matrix for all 1's, and $\mathrm{Nei}(i,j)=1$ if u_i neighbors u_j; 0 otherwise.
6. Λ_k denotes kth row of Λ_{ij}.

A *labeling* is a vector $L=(L_1,\cdots,L_n)'$, where $L_i=(l_{i1},\cdots,l_{im})$ in Λ_i is a Boolean vector with $l_{ij}=1$ if label λ_j is a possible label for object u_i; 0 otherwise.

A labeling is *consistent* if for every i and k

$$l_{ik}\leqslant\prod_{j=1}^{n}\left[\sum_{p=1}^{m}\left(l_{ik}*l_{jp}*\Lambda_{ij}(k,p)\right)\right]. \tag{1}$$

It can be rewritten as

$$l_{ik}\leqslant l_{ik}*\prod_{j=1}^{n}\left[\sum_{p=1}^{m}\left(l_{jp}*\Lambda_{ij}(k,p)\right)\right]. \tag{2}$$

If the l_{ik}'s, $k=1,m$ are now gathered together in vector form

$$\begin{bmatrix}l_{i1}\\l_{i2}\\\vdots\\l_{im}\end{bmatrix}\leqslant\begin{bmatrix}l_{i1}\\l_{i2}\\\vdots\\l_{im}\end{bmatrix}*\begin{bmatrix}\sum_{p=1}^{m}\left(l_{1p}*\Lambda_{i1}(1,p)\right)\\\sum_{p=1}^{m}\left(l_{1p}*\Lambda_{i1}(2,p)\right)\\\vdots\\\sum_{p=1}^{m}\left(l_{1p}*\Lambda_{i1}(m,p)\right)\end{bmatrix}\times*\cdots*\begin{bmatrix}\sum_{p=1}^{m}\left(l_{np}*\Lambda_{in}(1,p)\right)\\\sum_{p=1}^{m}\left(l_{np}*\Lambda_{in}(2,p)\right)\\\vdots\\\sum_{p=1}^{m}\left(l_{np}*\Lambda_{in}(m,p)\right)\end{bmatrix} \tag{3}$$

or

$$\begin{bmatrix}l_{i1}\\l_{i2}\\\vdots\\l_{im}\end{bmatrix}\leqslant\begin{bmatrix}l_{i1}\\l_{i2}\\\vdots\\l_{im}\end{bmatrix}*\begin{bmatrix}L_1*\Lambda_{i1}(1)^t\\L_1*\Lambda_{i1}(2)^t\\\vdots\\L_1*\Lambda_{i1}(m)^t\end{bmatrix}*\cdots*\begin{bmatrix}L_n*\Lambda_{1n}(1)^t\\L_n*\Lambda_{1n}(2)^t\\\vdots\\L_n*\Lambda_{1n}(m)^t\end{bmatrix} \tag{4}$$

or

$$L_i\leqslant L_i*\prod_{j=1}^{n}\left(\left\{\left[L_j\right]\times\left[\Lambda_{ij}(1)'\cdots\Lambda_{ij}(m)'\right]\right\}^t\right). \tag{5}$$

Let

$$P=\begin{bmatrix}P_1\\P_2\\P_3\\\vdots\\P_n\end{bmatrix} \tag{6}$$

where the column vector

$$P_i=\prod_{j=1}^{n}\left(\left\{\left[L_j\right]\times\left[\Lambda_{ij}(1)\cdots\Lambda_{ij}(m)\right]\right\}^t\right). \tag{7}$$

Gathering together the L_i's, $i=1,n$, we have

$$L\leqslant L*P. \tag{8}$$

This formulation emphasizes the relation to classical relaxation. The relaxation is achieved by repeating

$$L\Leftarrow L*P \tag{9}$$

until L does not change value.

B. An Example

Suppose that we are analyzing a picture of a scene, with the aim of describing it, and that we have detected a set of objects $u_1,\cdots,u_n$ in the scene, but have not identified them unambiguously. The relationships that exist among the objects are used to eliminate the ambiguity.

An example for eliminating the ambiguity in a region coloring problem is given here to demonstrate these ideas and computation procedures. For simplicity, consider the case of three regions to be colored red, green, or blue with the constraints:

1. Region 1 must be red.
2. Region 3 must be blue.
3. No two regions may be colored the same color.

Thus, $u_i=$ Region i (for $i=1,2,3$) and:

$$U=\{u_1,u_2,u_3\}$$

$$\Lambda=\{\lambda_1,\lambda_2,\lambda_3\}$$

where λ_1 is red, λ_2 is green, and λ_3 is blue. Since region 1 must be red, we have

$$\Lambda_1=[1\quad 0\quad 0]^t$$

and since region 3 must be blue:

$$\Lambda_3=[0\quad 0\quad 1]'.$$

Finally, since there is no restriction on region 2's color, we have all possibilities:

$$\Lambda_2=[1\quad 1\quad 1]'.$$

Since only similar colors are incompatible, we have

$$C=\begin{pmatrix}0&1&1\\1&0&1\\1&1&0\end{pmatrix} \tag{10}$$

for different objects, and

$$C=\begin{pmatrix}1&0&0\\0&1&0\\0&0&1\end{pmatrix} \tag{11}$$

for the same object.

We see then that C actually depends on the objects under consideration; i.e., technically, we should write C_{ij}, which is identified as

$$C_{ij}=\begin{pmatrix}\mathrm{Nei}(i,j)'&\mathrm{Nei}(i,j)&\mathrm{Nei}(i,j)\\\mathrm{Nei}(i,j)&\mathrm{Nei}(i,j)'&\mathrm{Nei}(i,j)\\\mathrm{Nei}(i,j)&\mathrm{Nei}(i,j)&\mathrm{Nei}(i,j)'\end{pmatrix} \tag{12}$$

where

$$\mathrm{Nei}(i,j)=\begin{cases}0, & \text{if Region } i \text{ does not neighbor Region } j\\1, & \text{if Region } i \text{ does neighbor Region } j.\end{cases}$$

Now we can calculate Λ_{ij} as

$$\Lambda_{11}=\left([1\quad 0\quad 0]'\times[1\quad 0\quad 0]\right)*((0'E)+C)=\begin{pmatrix}1&0&0\\0&0&0\\0&0&0\end{pmatrix}*\begin{pmatrix}1&0&0\\0&1&0\\0&0&1\end{pmatrix}=\begin{pmatrix}1&0&0\\0&0&0\\0&0&0\end{pmatrix} \tag{13}$$

$$\Lambda_{12}=\left([1\quad 0\quad 0]'\times[1\quad 1\quad 1]\right)*((1'E)+C)=\begin{pmatrix}1&1&1\\0&0&0\\0&0&0\end{pmatrix}*\begin{pmatrix}0&1&1\\1&0&1\\1&1&0\end{pmatrix}=\begin{pmatrix}0&1&1\\0&0&0\\0&0&0\end{pmatrix} \tag{14}$$

$$\Lambda_{13}=\left([1\quad 0\quad 0]'\times[0\quad 0\quad 1]\right)*((1'E)+C)=\begin{pmatrix}0&0&1\\0&0&0\\0&0&0\end{pmatrix}*\begin{pmatrix}0&1&1\\1&0&1\\1&1&0\end{pmatrix}=\begin{pmatrix}0&0&1\\0&0&0\\0&0&0\end{pmatrix} \tag{15}$$

$$\Lambda_{21}=\left([1\quad 1\quad 1]'\times[1\quad 0\quad 0]\right)*((1'E)+C)=\begin{pmatrix}1&0&0\\1&0&0\\1&0&0\end{pmatrix}*\begin{pmatrix}0&1&1\\1&0&1\\1&1&0\end{pmatrix}=\begin{pmatrix}0&0&0\\1&0&0\\1&0&0\end{pmatrix} \tag{16}$$

$$\Lambda_{22}=\left([1\quad 1\quad 1]'\times[1\quad 1\quad 1]\right)*((0'E)+C)=\begin{pmatrix}1&1&1\\1&1&1\\1&1&1\end{pmatrix}*\begin{pmatrix}1&0&0\\0&1&0\\0&0&1\end{pmatrix}=\begin{pmatrix}1&0&0\\0&1&0\\0&0&1\end{pmatrix} \tag{17}$$

$$\Lambda_{23}=\left([1\quad 1\quad 1]'\times[0\quad 0\quad 1]\right)*((1'E)+C)=\begin{pmatrix}0&0&1\\0&0&1\\0&0&1\end{pmatrix}*\begin{pmatrix}0&1&1\\1&0&1\\1&1&0\end{pmatrix}=\begin{pmatrix}0&0&1\\0&0&1\\0&0&0\end{pmatrix} \tag{18}$$

$$\Lambda_{31}=\left([0\quad 0\quad 1]'\times[1\quad 0\quad 0]\right)*((1'E)+C)=\begin{pmatrix}0&0&0\\0&0&0\\1&0&0\end{pmatrix}*\begin{pmatrix}0&1&1\\1&0&1\\1&1&0\end{pmatrix}=\begin{pmatrix}0&0&0\\0&0&0\\1&0&0\end{pmatrix} \tag{19}$$

$$\Lambda_{32}=\left([0\quad 0\quad 1]'\times[1\quad 1\quad 1]\right)*((1'E)+C)=\begin{pmatrix}0&0&0\\0&0&0\\1&1&1\end{pmatrix}*\begin{pmatrix}0&1&1\\1&0&1\\1&1&0\end{pmatrix}=\begin{pmatrix}0&0&0\\0&0&0\\1&1&0\end{pmatrix} \tag{20}$$

$$\Lambda_{33}=\left([0\quad 0\quad 1]'\times[0\quad 0\quad 1]\right)*((0'E)+C)=\begin{pmatrix}0&0&0\\0&0&0\\0&0&1\end{pmatrix}*\begin{pmatrix}1&0&0\\0&1&0\\0&0&1\end{pmatrix}=\begin{pmatrix}0&0&0\\0&0&0\\0&0&1\end{pmatrix}. \tag{21}$$

The (p,q) entry of Λ_{ij} tells if λ_p at object i is compatible with λ_q at object j. For example, Λ_{11} reveals that only λ_1 is compatible with λ_1 at object 1; i.e., that Region 1 must be colored red.

Finally, we continue the iteration process. According to (2), for $i=1$ and $k=1$:

$$\begin{aligned}l_{11}^{(n)}&\leqslant l_{11}^{(n-1)}\\&*\left[l_{11}^{(n-1)}*\Lambda_{11}(1,1)+l_{12}^{(n-1)}*\Lambda_{11}(1,2)+l_{13}^{(n-1)}*\Lambda_{11}(1,3)\right]\\&*\left[l_{21}^{(n-1)}*\Lambda_{12}(1,1)+l_{22}^{(n-1)}*\Lambda_{12}(1,2)+l_{23}^{(n-1)}*\Lambda_{12}(1,3)\right]\\&*\left[l_{31}^{(n-1)}*\Lambda_{13}(1,1)+l_{32}^{(n-1)}*\Lambda_{13}(1,2)+l_{33}^{(n-1)}*\Lambda_{13}(1,3)\right]\end{aligned} \tag{22}$$

$$\begin{aligned}1&\leqslant 1*[1*1+0*0+0*0]\\&\quad*[0*0+0*0+1*1]\\&\quad*[1*0+1*1+1*1]\\1&\leqslant 1*1\\1&\leqslant 1 \text{ which is true.}\end{aligned}$$

This says that the color red is all right for Region 1. To determine if the color red is possible for Region 2, we must find l_{21}. For $i=2$ and $k=1$:

$$\begin{aligned}l_{21}^{(n)}&\leqslant l_{21}^{(n-1)}\\&*\left[l_{11}^{(n-1)}*\Lambda_{21}(1,1)+l_{12}^{(n-1)}*\Lambda_{21}(1,2)+l_{13}^{(n-1)}*\Lambda_{21}(1,3)\right]\\&*\left[l_{21}^{(n-1)}*\Lambda_{22}(1,1)+l_{22}^{(n-1)}*\Lambda_{22}(1,2)+l_{23}^{(n-1)}*\Lambda_{22}(1,3)\right]\\&*\left[l_{31}^{(n-1)}*\Lambda_{23}(1,1)+l_{32}^{(n-1)}*\Lambda_{23}(1,2)+l_{33}^{(n-1)}*\Lambda_{23}(1,3)\right]\end{aligned} \tag{23}$$

$$\begin{aligned}1&\leqslant 1*[1*0+0*0+0*0]\\&\quad*[1*1+1*0+1*0]\\&\quad*[0*0+0*0+1*1]\\1&\leqslant 1*0\\1&\leqslant 0 \text{ which is false.}\end{aligned}$$

Thus, l_{21} must be set to zero. Likewise, for $i=2$ and $k=3$, l_{23} is set to zero, and blue is not a possible label for Region 2. Finally, for $i=2$ and $k=2$:

$$\begin{aligned} l_{22}^{(n)} &\leqslant l_{22}^{(n-1)} \\ &\quad *\left[l_{11}^{(n-1)} * \Lambda_{21}(2,1) + l_{12}^{(n-1)} * \Lambda_{21}(2,2) + l_{13}^{(n-1)} * \Lambda_{21}(2,3)\right] \\ &\quad *\left[l_{21}^{(n-1)} * \Lambda_{22}(2,1) + l_{22}^{(n-1)} * \Lambda_{22}(2,2) + l_{23}^{(n-1)} * \Lambda_{22}(2,3)\right] \\ &\quad *\left[l_{31}^{(n-1)} * \Lambda_{23}(2,1) + l_{32}^{(n-1)} * \Lambda_{23}(2,2) + l_{33}^{(n-1)} * \Lambda_{23}(2,3)\right] \end{aligned} \tag{24}$$

$$\begin{aligned} 1 &\leqslant 1*[1*1+0*0+0*0] \\ &\quad *[1*0+1*1+1*0] \\ &\quad *[0*0+0*0+1*1] \\ 1 &\leqslant 1 \text{ which is true.} \end{aligned}$$

We see then that the values of l_{11}, l_{22}, and l_{33} are not affected by the change of l_{21} and l_{23} to zero. In fact, the system of equations stabilizes after the change of l_{21} and l_{23}, and the result is $l_{11}=l_{22}=l_{33}=1$, while all other hypotheses are zero. Thus, the only consistent labeling is to label Regions 1, 2, and 3 the colors red, green, and blue, respectively.

III. The Hardware Implementation Problem for the DRA

To simplify our prototype chip designs, the following assumptions are adopted in our implementations. First, since the design is allowed to be specified for arbitrary numbers of objects and labels as long as the chip size permits, we assume these two numbers are equal, i.e., $n=m$. Secondly, for practical real image processing, we have chosen $n=m=8$. It is clearly indicated that these assumptions are reasonable and meaningful, without losing any generality for developing advanced general purpose DRA architectures [7], [8].

The problem of *DRA Hardware Implementation* has been defined as finding out the *labeling matrix L*:

$$L=(L_1, L_2, \cdots, L_i, \cdots, L_n)^t = \begin{pmatrix} l_{11}, l_{12}, \cdots, l_{1n} \\ l_{21}, l_{22}, \cdots, l_{2n} \\ \cdots\cdots\cdots \\ l_{n1}, l_{n2}, \cdots, l_{nn} \end{pmatrix} \tag{25}$$

for the given *world model*, given the *initial labeling matrix*:

$$\Lambda=(\Lambda_1, \Lambda_2, \cdots, \Lambda_i, \cdots, \Lambda_n)^t = \begin{pmatrix} l_{11}^{(o)}, l_{12}^{(o)}, \cdots, l_{1n}^{(o)} \\ l_{21}^{(o)}, l_{22}^{(o)}, \cdots, l_{2n}^{(o)} \\ \cdots\cdots\cdots \\ l_{n1}^{(o)}, l_{n2}^{(o)}, \cdots, l_{nn}^{(o)} \end{pmatrix} \tag{26}$$

and the *object compatibility matrices* C_{ij}, of (12), for every i and j $(i, j=1,2,\cdots,n)$.

IV. A Parallel Tree-Structured Reformulation for the DRA

A. A Conventional Design and its Complexity Analysis

A conventional hardware design DRA1 for an 8-label 8-object DRA problem is presented in [7] and [15]. The computational strategy used in that design is to serially compute each intermediate element of matrices $\Lambda_{ij}(p,q)$ and l_{ij} and periodically read and write L, Λ, $\Lambda_{ij}(p,q)$ and C_{ij} from and into memories. Since the computation mechanism imbedded in this design is purely an *I/O bounded* computation, the upper bound of execution time is on the order of hours for a 3-μm NMOS process. Finally, the complete system takes three separate chips (totally about 80,000 transistors). This design has revealed the inherent computation complexity for DRA's hardware implementation.

Referring to (1)–(7), in order to store the initial labels Λ, matrices $C_{ij}(k,p)$ and the intermediate results of all elements of matrixes $\Lambda_{ij}(p,q)$ $(i,j,p,q=1,\cdots,n)$, the space complexity is on the order of

$$O(2n^2+3n^4)=O(n^4). \tag{27}$$

For practical applications, the label number could be 8, 16, or 32; thus, the bit memory requirements for these different cases are 12K, 48K, and 192K, respectively. As shown in design [15], this adds to the circuit size and is a bottleneck when n is large.

The time complexity can be estimated from (22)–(24). During each iteration, at least $2\times4\times n^4\times n$ read and write memory operations will need to be performed. Assuming $t_{\text{read}} \approx t_{\text{write}} \approx 500$ ns for an NMOS process, the computation time complexity of each iteration is $O(n^5)$ (assuming the assumption that the unit time is 500 ns). Multiplying the worst-case iteration times $O(ea^3)$ [5], which is on the order of $O(n^5)$ and is determined by the feature of the computational model, the execution is terribly slow.

B. A Highly Parallel Tree-Structured Reformulation for the DRA

It should be clear that any attempt to speed up an I/O-bound computation must rely on an increase in the memory bandwidth. Speeding up a compute-bound computation, however, may frequently come from the concurrent use of many processing elements. The degree of parallelism in a special-purpose system is largely determined by the underlying algorithm. In order to solve the complexity met in the conventional DRA1 design, the following three steps have been taken to design a hardware algorithm that supports a high degree of concurrency in the DRA computation.

1) Constructing the Parallel Computation Tree: When more effort is spent on analyzing (2), we see that element $\Lambda_{ij}(k,p)$ can be decomposed as

$$\Lambda_{ij}^{(n-1)}(k,p)=l_{ik}^{(o)} l_{jp}^{(o)} C_{ij}(k,p) \tag{28}$$

which can form a leaf node as shown in Fig. 1 so that (2) can be hierarchically formed as a tree-like structure with

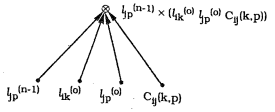

Fig. 1. Leaf node for computing $l_{jp} \times \Lambda_{ij}(k,p)$.

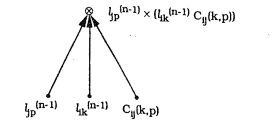

Fig. 2. Modified leaf node computation for $l_{jp} \times \Lambda_{ij}(k,p)$.

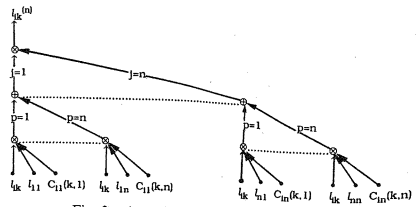

Fig. 3. A parallel tree for computing nth l_{ik}.

each level imbedded in the parallel computation for their leaves' operands, as shown in Fig. 3.

2) Speeding Up the Iteration: The node computation in Fig. 1 can be speeded up by replacing the initial $l_{ik}^{(o)}$ and $l_{jp}^{(o)}$ elements with the $n-1$th iterated results $l_{ik}^{(n-1)}$ and $l_{jp}^{(n-1)}$. The modified computation composed of the leaf node is shown in Fig. 2. The computation tree for $l_{ik}^{(n)}$ is formed as shown in Fig. 3.

In Section V, it is shown that the bottom-most leaves' operands have been associated with a data pipelining channel, completing a tree-root pipelining scheme which supports a highly concurrent SIMD computation for image relaxation operations.

3) Introducing Time Dimension in Computation: To compute an n-object n-label relaxation problem, the total number n^2 of l_{ik}'s need to be evaluted. This means at least 64 computation trees as shown in Fig. 3 need to be built inside the circuit, which greatly increases the circuit size. To minimize this problem, each operand at the bottom of the tree has been constructed in a time dimension. As the time changes, different l_{ik}'s $(i=1,\cdots,n)$ can be generated. This computation philosophy does not add more time complexity but decreases the computation tree requirement to n [11]. The introduction of the time dimension constitutes the theoretical basis for recursive computation [10] and interleaved processing.

The parallel tree-structured reformulation and tree-root pipelining for the DRA take advantage of a high degree of pipelining and multiprocessing. It gets rid of the need to store and compute each element in the compatibility matrix

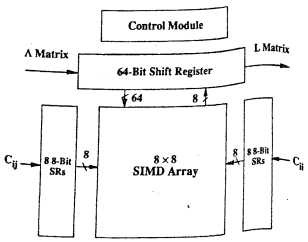

Fig. 4. Circuit block diagram for DRA2 architecture.

for u_i and u_j, i.e., (13)–(21), decreasing both the space and time complexities to $O(n^2)$. Thus, the DRA2 design eliminates two 4K memories from the original DRA1 design, only a 64-bit shift register is required to store 64 intermediate label elements. Since each computation takes 64 cycles, assuming a clock cycle is about 120 ns (in 3-μm NMOS process, PPL design methodology) and the maximum possible iteration time is $O(n^5)$, the maximum possible execution time given is within milliseconds.

V. Implementation Issues for the DRA2 Architecture

A. Basic Principles and Implementation Strategies for DRA2 Circuit

1) System Architecture and Block Diagram: The block diagram of the DRA2 architecture is illustrated in Fig. 4. The chip consists of four functional blocks.

1. *Compatibility matrix registers (CMR)*: C_{ij} Registers are a set of eight 8-bit shift registers in the leftmost part of the circuit; they are used for storing each C_{ij} matrix. Another set of C_{ii} registers in the rightmost part of the circuit are for storing C_{ii}.
2. *8×8 SIMD multiprocessor array (SMA)*: The SIMD arrray is composed of 8×8 simple and regular cells. They are predefined to map the parallel computation tree of Fig. 3 into silicon. A number of horizontal and vertical communication wires are designed around the four edges of the cells to make use of higher degrees of parallelism in computation.
3. *L-matrix shift register (LSR)*: It is used for 1) the input and output data paths for original and final labeling matrices, 2) the pipelining channel for tree-root operands broadcasting and pipelining, forming a recursive DRA computational wavefront, and 3) performing temporarily the data storing and updating.
4. *Control Module (CM)*: This module includes four units. An *8-bit comparator* is located on top of the first 8-bit shift register of the LSR to sense the equality between the nth output vector $L_i^{(n)}$ of the SIMD array and the corresponding n-1th row

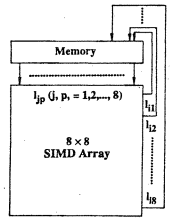

Fig. 5. Basic principle of the SIMD DRA2 architecture.

vector $L_i^{(n-1)}$ inside the LSR. A *timer* serves as both the cyclic pacer and tagged-bit signal generator for iteration control. An *8-bit state register* is used for collecting comparison results from the comparator and monitoring iteration states. Finally, a *finite state machine* (FSM) is built for performing a self-timed synchronization among these functional blocks and host computer.

The diagram of Fig. 4 of four functional blocks also serves as the PPL layout floorplan for efficient layout (in Section V-B).

2) SIMD Array and Its Cell Design: The basic principle of the SIMD DRA2 architecture for DRA2 is illustrated in Fig. 5. By replacing a single processing element (PE) with an array of 8×8 PE's, a higher computation throughput can be achieved without increasing memory bandwidth. The function of the memory (i.e., the L matrix shift registers) in the diagram is to pulse data l_{jp} $(j, p = 1,2,\cdots,n)$ through the array of cells. Then new data l_{ik} $(i,k = 1,2,\cdots,n)$ are returned to memory in a rhythmic fasion. The crux of this approach is to ensure that once the data are brought out from the memory they can be used effectively at each cell they pass while being pumped through the entire array.

To perform the parallel DRA2 computation, two cells (as illustrated in Fig. 6(a) and (b)) with almost identical logic and structure were used in constructing the entire array. The only difference is that the first cell performs the generation of the broadcasting signals for each row array. The construction of the SIMD array using these two cells is illustrated in Fig. 7. In Fig. 6(a)

$$b_k = l_{ik}, \qquad \text{at column } j=1. \tag{29}$$

$$\begin{aligned}\text{Out}(j,k)_{j=1} &= \sum_{p=1}^{n} \left(l_{jp} \times \Lambda_{ij}(k,p)\right) \\ &= \sum_{p=1}^{n} \left(l_{jp} \times l_{ik} \times C_{ii}(k,p)\right) \\ &= \sum_{p=1}^{n} \left(l_{jp} \times b_k \times C_{ii}(k,p)\right) \\ &= \sum_{p=1}^{n} \overline{\left(\overline{l_{jp}} + \overline{b_k} + \overline{C_{ij}}\right)}. \end{aligned} \tag{30}$$

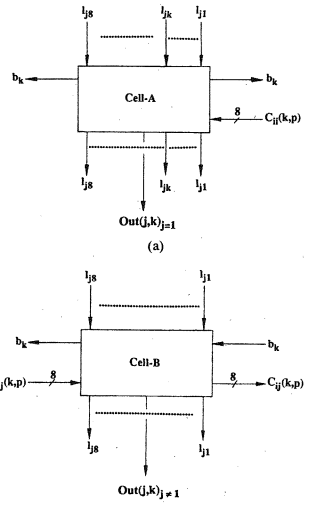

Fig. 6. Two cells in DRA2 SIMD array.

In Fig. 6(b)

$$b_{k\,(\text{in})} = b_{k\,(\text{out})}, \qquad \text{at column } j \neq 1. \tag{31}$$

$$\begin{aligned}\text{Out}(j,k)_{j\neq 1} &= \sum_{p=1}^{n} \left(l_{jp} \times \Lambda_{ij}(k,p)\right) \\ &= \sum_{p=1}^{n} \left(l_{jp} \times l_{ik} \times C_{ii}(k,p)\right) \\ &= \sum_{p=1}^{n} \left(l_{jp} \times b_k \times C_{ii}(k,p)\right) \\ &= \sum_{p=1}^{n} \overline{\left(\overline{l_{jp}} + \overline{b_k} + \overline{C_{ij}}\right)}. \end{aligned} \tag{32}$$

According to Fig. 3 and (29)–(32), these two cells are implemented in two levels of NOR gate combinational logic. Their PPL [12], [13] layouts can be easily identified in Fig. 12.

3) Circuit Features and Design Techniques: In addition to designing the simple and regular cells, several efficient techniques (such as interleaved processing, multiple signal broadcasting, and self-timed synchronization) were applied to the implementation of the SIMD DRA2 architecture.

a) Recursive computation and interleaved processing: Since the introduction of the time dimension in Section IV, the SIMD array in Fig. 7 possesses a time-varying characteristic which makes recursive computation and interleaved processing possible. Let's focus on the first column $(j=1)$ array. It is clearly indicated that the first input vector, which is the ith row vector of L, the labeling

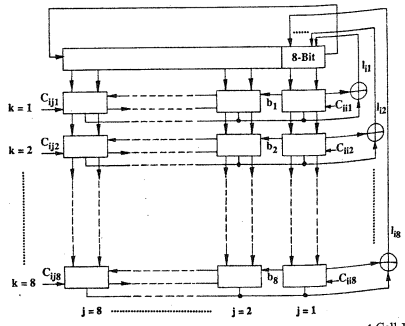

Fig. 7. Construction of SIMD DRA2 array using Cell-A and Cell-B.

matrix, at the n-1th iteration, is fed into the first column of DRA2 array as

$$(l_{j1}, l_{j2}, l_{j3}, \cdots, l_{jn})$$

for $j=1,\cdots,n$. The corresponding output vector L_i of the SIMD array, which is the ith row vector of the L labeling matrix at the nth iteration, is generated

$$(l_{i1}, l_{i2}, l_{i3}, \cdots, l_{in})$$

where i is fixed at a time $t=i$. As time moves forward, the elements in the L shift register have shifted from the left to the right in an 8-clock-pace fashion. The time-varying feature of the entire DRA2 array can best be described by the following two *topological index equations*:

$$\begin{aligned} i &\leftarrow i + t \bmod n \\ j &\leftarrow j + t \bmod n. \end{aligned}$$

For example, in the DRA2 system, at time $t=i=1$, vector L_1

$$(l_{11}, l_{12}, l_{13}, l_{14}, l_{15}, l_{16}, l_{17}, l_{18})$$

is generated, and at time $t=i=2$, vector L_2

$$(l_{21}, l_{22}, l_{23}, l_{24}, l_{25}, l_{26}, l_{27}, l_{28})$$

is generated, etc. (See Fig. 8.) Each L_i vector is computed based on the interleaved utilization of the SIMD array, whereas eight L_i vectors form an entire computational wavefront of the L labeling matrix, of the nth relaxation iteration. Note that we use the number of n computing trees for generating n^2 l_{ik}'s in $O(n^2)$ time, we may also use the same number of computing trees to compute the same number of l_{ik}'s in $O(n)$ time, based on the strategy of dynamically configuring the DRA architectural wavefront [8].

b) Multiple signal broadcasting: The broadcasting technique is probably one of the most obvious ways to make multiple use of each input element. It plays an important

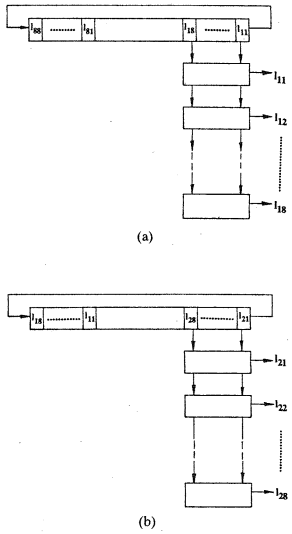

Fig. 8. Computational wavefront pipelining and circulation for interleaved processing.

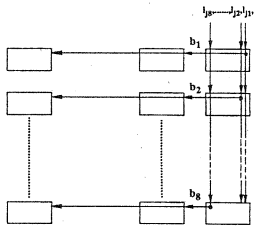

Fig. 9. Broadcasting scheme for b_k signal.

role in making the parallel computation tree of Fig. 3 implementable. Two multiple broadcasting schemes are used in DRA2 architecture. In the first, n^2 vertical broadcasting lines from each pipelining operand are connected to the bottom most leaves' node of each parallel computing tree. Secondly, as depicted in Figs. 6 and 9, Cell-A at column $j=1$ is used to jog signal $l_{ik}^{(n-1)}$ (which is the nth l_{jp}) and then propagate it horizontally from right to left through the entire row array. Thus, the output vector of the SIMD array, i.e., $(l_{i1}, l_{i2}, l_{i3}, l_{i4}, l_{i5}, l_{i6}, l_{i7}, l_{i8})$, can be generated simultaneously in a highly concurrent manner.

c) Self-timed synchronization and tagged-bit control: By using recursive computation and interleaved processing, the computational task has been decomposed into the smallest computing piece L_i. To compute each vector L_i, the globally synchronized SIMD array of Fig. 7 is used.

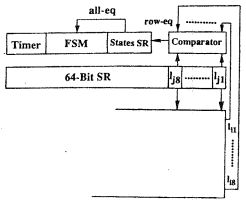

Fig. 10. Self-timed synchronization.

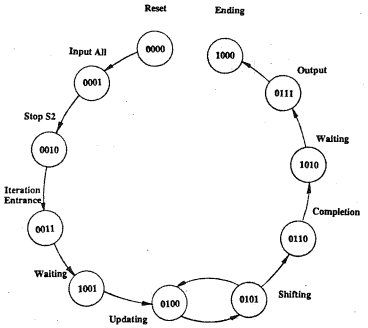

Fig. 11. State graph of the finite state machine.

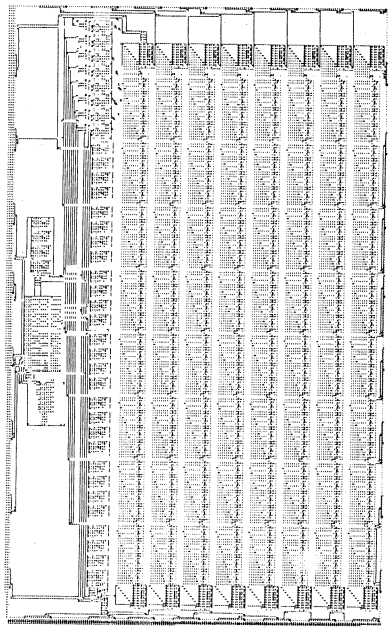

Fig. 12. The PPL layout of the SIMD DRA2 architecture.

For completing the entire relaxation computation, this synchronous array is imbedded into a self-timed system. The self-timed asynchronous scheme may be costly in terms of extra hardware and delay in each element, but it has the advantage that the time required for a communication event between two elements is independent of the size of the entire system. Also, it is easy to design and validate a self-timed state machine in PPL methodology [9].

Among the 64-bit L shift registers, the rightmost first 8-bit SR is one which is able to parallel load in the nth output vector from the SIMD array in order to update the current n-1th L_i row vector. This iteration and updating process is the core of the relaxation process described in (8). To sense the completion of computation, a comparator is built on top of the first 8-bit SR. If two vectors are equal, a *row-eq* signal of 1 is produced and stored into *8-bit states SR* of the control module; otherwise, a 0 signal is sent. As soon as the state register gets eight 1's, which means the equality of (8) is reached, an *all-eq* signal is issued to the FSM. Since the control processes in this system are based on the data validity of a *control data flow*, a reliable and fast execution in a data-driven environment is created. The control mechanism used in the parallel DRA2 architecture is shown in Fig. 10.

To ensure that the iteration cycle completes at the end of n^2 cycles, a *tagged-bit* is derived from an ANDed term of both the l_{11} bit and the 64th-count of the timer, which has served as a reliable alignment signal for computation in the control flow.

The state graph of the FSM is illustrated in Fig. 11.

B. PPL Layout

The DRA2 chip was built by assembling the four functional blocks in Fig. 4 using path programmable logic (PPL) tools at the University of Utah [12], [13]. Since parallel computation and the SIMD array greatly simplify the design difficulties, the PPL layout is very simple and straightforward. An overview of the complete PPL layout, which is a PPL mapping of the block *floor plan* in Fig. 4, is shown in Figure 12. The photomicrograph of the fabricated DRA2 chip is in Fig. 13.

For details related to the complete system design, the simulation results, interfacing strategy with host computer, timing and wiring delay analysis, testing, pinout description, and fabrication of the DRA2 chip see [7] and [14].

C. Simulation

1) Functional Simulation: Functional simulation is aimed at the verification of the correctness of the algorithm and

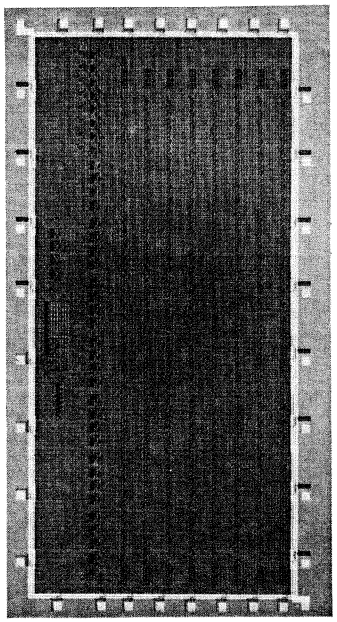

Fig. 13. The photomicrograph of the DRA2 chip.

data structure, and the discovery of limitations and problems which may occur during practical implementation. A number of high-level functional simulations were performed during the formulation of DRA.

2) Logical Simulation: Logic simulation for the DRA2 chip was performed for individual modules (and the entire circuit as well) using the PPL topological circuit simulation tool *SIMPPL*. Simulation using SIMPPL is performed by assigning logical values to every node in the circuit. Input values are assigned and allowed to ripple through the circuit. Output values are then checked to ensure that the correct values are produced. For detailed functionality and usage about PPL simulation tools see [12] and [13].

An 8-object, 8-label coloring identification problem was selected for logical simulation as shown in the following. The input to the circuit includes matrixes C_{ii}, C_{ij}, and $L^{(o)}$. Matrices C_{ii} and C_{ij} are the inherent label pair's relationships. Suppose $L_{ij}^{(o)}$ are the raw data with ambiguity detected by a robot observer. The first seven initial labels $(L_1, L_2, L_3, L_4, L_5, L_6, L_7)$ of $L^{(o)}$ indicate seven distinct regions of color on an object, the 8th label (L_8 = 11111111) means the color in the eighth region is a mixture of all eight colors in these eight areas. We frequently meet such a situation. For example, suppose an airplane is flying; its major parts are clear, but one area in its body is blurred due to the plane's motion. Therefore,

$$C_{ii} = \begin{pmatrix} 1&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0\\ 0&0&1&0&0&0&0&0\\ 0&0&0&1&0&0&0&0\\ 0&0&0&0&1&0&0&0\\ 0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0&1 \end{pmatrix}$$

$$C_{ij} = \begin{pmatrix} 0&1&1&1&1&1&1&1\\ 1&0&1&1&1&1&1&1\\ 1&1&0&1&1&1&1&1\\ 1&1&1&0&1&1&1&1\\ 1&1&1&1&0&1&1&1\\ 1&1&1&1&1&0&1&1\\ 1&1&1&1&1&1&0&1\\ 1&1&1&1&1&1&1&0 \end{pmatrix}$$

$$L^{(o)} = (L_1, L_2, L_3, L_4, L_5, L_6, L_7, L_8)^t$$

$$= \begin{pmatrix} 1&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0\\ 0&0&1&0&0&0&0&0\\ 0&0&0&1&0&0&0&0\\ 0&0&0&0&1&0&0&0\\ 0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&1&0\\ 1&1&1&1&1&1&1&1 \end{pmatrix}.$$

After relaxation computation, the eighth color in that region has been identified as $L_8 = 00000001$ in matrix $L^{(n)}$. The simulation file for these inputs is given in [7]; the final labeling matrix sought is

$$L^{(n)} = (L_1, L_2, L_3, L_4, L_5, L_6, L_7, L_8)^t$$

$$= \begin{pmatrix} 1&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0\\ 0&0&1&0&0&0&0&0\\ 0&0&0&1&0&0&0&0\\ 0&0&0&0&1&0&0&0\\ 0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0&1 \end{pmatrix}$$

VI. Comparison with the DRA1 Design

A brief comparison with the parallel DRA2 architecture and our first DRA1 design for the same DRA problem has been summarized in Fig. 14.

VII. Conclusions

The parallel tree-structured reformulation and multiple tree-root pipelining scheme broke through the bottleneck in the DRA hardware design. Compared to the first conventional design (DRA1), the DRA2 design achieves a very impressive performance in terms of speed, size, and memory access requirements. The DRA2 chip has been fabricated using a 3-μm NMOS technology. We hope to test it soon; the maximum expected clock speed is around 5 MHz.

Complexity Comparisons

	DRA1	DRA2
Hardware Algorithm	Serial Computation	Highly Concurrent
Space Complexity	$O(n^4)$	$O(n^2)$
Memory Requirement	12 K	None
Time Complexity	$O(n^5)$	$O(n^2)$
Computation Time	Hours	Microsecond to Milliseconds
Layout	Irregular and Hard	Regular and Simple
Control Strategy	Complicated	Simple
Entire System	Three Chips (DRA1 chip plus External 4-K RAM and Bus Ctl Chip)	One Chip
# of Transistors	~ 80,000	6,382

DRA Chip Descriptions

	DRA1	DRA2
# of Transistors	39,502	6,382
Pins	64	35
Sizes	300 × 250	181 × 249
(PPL row & column)		75 × 75 (Using Full-Custom Designed Cell-A and Cell-B in SIMD Array)

Fig. 14. The comparisons with the DRA1 design.

The implementation of DRA2 architecture has paved the way for developing various kinds of fast, general purpose, and high-performance discrete relaxation architectures for real-time digital image processing. Currently, several advanced developments for those DRA architectures using an 1.2-μm CMOS technology are being designed and the clock speed is expected to be over 50 MHz. Further research in this area will allow us to imbed the highest degree of flexibility in DRA design by allowing programmability in cells, as well as reconfigurability of cell interconnections, for generating efficient and configurable DRA computation architectures.

Acknowledgment

The authors would like to thank D. C-L. Ku at Stanford University for his DRA1 system design.

References

[1] A. Rosenfeld, R. A. Hummel, and S. W. Zucker, "Scene labeling by relaxation operations," *IEEE Trans. Syst. Man. Cyber.*, vol. SMC-6, pp. 420–433, June 1976.
[2] D. H. Ballard and C. M. Brown, *Computer Vision*. Englewood Cliffs, NJ: Prentice-Hall, 1982.
[3] R. V. Southwell, *Relaxation Methods in Engineering Science*. London: Oxford University Press, 1940.
[4] D. Waltz, "Understanding line drawings of scenes with shadows," in *Psychology of Computer Vision*, P. H. Winston, Ed. New York: McGraw-Hill, 1975, pp. 19–91.
[5] T. C. Henderson, "A note on discrete relaxation," *Computer Vision, Graphics and Image Processing*, vol. 28, pp. 384–388, 1984.
[6] T. C. Henderson, *Discrete Relaxation*. London: Oxford University Press, to appear.
[7] W. Wang, J. Gu, and T. C. Henderson, "A systolic array implementation of a discrete relaxation algorithm," Tech. Rep. UUCS-TR-86-008, Dept. Computer Science, Univ. of Utah, Mar. 1986.
[8] J. Gu, W. Wang, and T. C. Henderson, "An $O(n)$ time discrete relaxation architecture for real-time processing of the consistent labeling problem," Res. Tech. Rep. UUCS-TR-86-116, Dept. Computer Science, Univ. of Utah, Dec. 1986.
[9] A. B. Hayes, "Self-timed IC design with PPL's," in *Proc. Third CalTech Conf. on VLSI*, Jan. 1983, pp. 257–273.
[10] U. Weiser and A. Davis, "A wavefront notation tool for VLSI array design," in *Proc. CMU Conf. on VLSI System and Computations*, Oct. 1981, pp. 226–234.
[11] K. Hwang and F. A. Briggs, *Computer Architecture and Parallel Processing*. New York: McGraw-Hill, 1984.
[12] K. F. Smith, T. M. Carter, and C. E. Hunt, "Structured logic design of integrated circuits using the storage/logic array (SLA)," *IEEE Trans. Electron Devices*, vol. ED-29, Apr. 1982.
[13] K. F. Smith, "PPL manual," Tech. Rep. UUCS-86-003, Dept. Computer Science, Univ. of Utah, Jan. 1986.
[14] MOSIS, "DRA2 chip fabrication report," Information Science Institute, Univ. of Southern California, Dec. 1986.
[15] D. Ku, "DRA1 chip implementation report," Project Report, Univ. of Utah, Mar. 1986.

Wei Wang received the B.S. degree in electrical engineering and computer science from the Hofei Institute of Technology, Hofei, P.R. China, in 1982.

She worked as a member of the technical staff at the National Institute of Electrical and Mechanical Engineering, Beijing, from 1982 to 1984. In 1984, she joined the faculty of the Computer Science Department, at the University of Science and Technology of China (USTC). Currently, she is a graduate student in the Electrical Engineering Department at the University of Utah. Her current research interests are in the areas of communication system, coding theory, digital signal processing, and VLSI architecture.

Jun Gu (S'86) received the B.S. degree in electrical engineering from the University of Science and Technology of China (USTC), Hofei, P.R. China, in 1982.

He worked as a faculty member from 1982 to 1983 in the Electrical Engineering Department, USTC. During 1984 and 1985, he pursued an M.S. degree on the topic of bioengineering and optical information processing in the Electrical Engineering Department at the University of Utah. Since September 1985, he has been a graduate student of Computer Science at the University of Utah, where he is currently a Ph.D. candidate. He was supported by an ASC Fellowship during 1984/85 and 1985/86, a University of Utah Research Fellowship during 1986/87, and this year has been awarded an ACM/IEEE scholarship. His current research interests include AI and robot vision, computer architecture, fault tolerant computing, design automation, and semiconductor devices.

Mr. Gu is a student member of the Association for Computing Machinery, the American Association for Artificial Intelligence, and a member of Sigma Xi.

Thomas C. Henderson (M'80–SM86') received the B.A. degree (with honors) in mathematics from Louisiana State University, in 1973, and the Ph.D. degree in computer science from the University of Texas at Austin in 1979.

He worked at DFVLR in Germany during 1980, and was at INRIA in France in 1981. He joined the Department of Computer Science at the University of Utah in 1982, where he is presently Associate Professor and Associate Chairman. Current areas of technical interest include multisensor systems, artificial intelligence, computer vision, architectures for robotics, and CAD-Based robotics.

Image Processing for Higher Definition Television

GARY J. TONGE

(*Invited Paper*)

Abstract **— The paper discusses the application of digital image-processing techniques to broadcast television with the goal of picture quality improvement. After defining picture quality targets and levels of system compatibility, a review of techniques is presented. These fall into the categories of achieving a wider picture format, improving vertical and horizontal resolution, and improving accurate color reproduction. Specific algorithms for vertical resolution improvement by display scan conversion and horizontal resolution improvement by three-dimensional signal processing are described. Many other approaches are also summarized and referenced, with a particular emphasis on European work.**

I. Introduction

OF THE MANY developments in digital image processing, one which will have an impact on most of the population, is in the area of consumer television. Digital storage and processing are already finding their way into consumer television equipment (teletext memories, picture-in-picture facilities, etc.), and the future promises much more. This paper addresses the increasing amount of research and development with the goal of higher definition television ("Hi-fi TV"). Some of the image processing being considered is of a complexity which is currently hard to imagine in the consumer environment. Nevertheless, the prospect of VLSI volume production makes it possible to consider some of the relatively complex approaches, provided that the improvement is worthwhile.

II. Picture Quality Targets

In a very general sense, the target is to produce a picture presentation in the home which is a sufficient improvement over the current norm to justify the extra expense. Typically, the improved quality will not be exploited to give a better picture definition as seen by the eye (at a typical viewing distance) but rather to enable a larger screen presentation.

Subjective tests in Japan [1] and in Europe [2] have shown that a larger viewing angle can increase the sense of involvement in a televised scene. Given that domestic television viewing disances are typically set more by practical constraints (room size, furniture, etc.) than by technical, this implies an increased screen size. The Japanese work also highlighted the importance of having a wider aspect ratio for large-screen television.

More specifically, targets for "high-definition television" (HDTV) have been set as an improvement in comparison with conventional television systems[1] in both vertical and horizontal resolution of about 2:1 in conjunction with a wider aspect ratio of around 5:3 [3]. This implies, for the same picture definition as seen by the eye at a fixed viewing distance, an increase in screen height by a factor of about 2 and screen width by a factor of around $2\frac{1}{2}$, as shown in Fig. 1.

The image-processing techniques described here can be used to achieve a broad range in the degree of improvement over conventional systems. The specific targets discussed above represent a tangible benchmark of quality which has been called "HDTV." It is not clear, however, what reduction in picture quality is necessary before an image ceases to be HDTV. With conventional television standards, for example, a domestic VCR provides a degraded picture. Nevertheless, from a users' point of view, this does not imply a destruction of the service. For this reason, it can be argued that the distinctive feature of "HDTV" is the wider aspect ratio format while the picture quality criteria are vague. Nevertheless, any improvements which can be provided by image-processing techniques in a cost-effective way are likely to find definite application, and an increasing future trend toward higher quality consumer television can be predicted.

III. Compatibility

Image-processing techniques for picture quality improvement can be applied in a number of different ways. At one extreme, they can be applied retrospectively to current TV services by processing in the receiver alone. At the other extreme, they can be particular to a proposed new completely different TV service. In the technical description which follows, the techniques are subdivided into four categories of "compatibility" with current systems.

Manuscript received February 14, 1987; revised June 11, 1987.
The author is with the Independent Broadcasting Authority, Crawley Court, Winchester, Hants, England.
IEEE Log Number 8716563.

[1] Recent standardization efforts for HDTV studio production have been based on these targets with respect to the currently agreed digital *studio* television standard. This implies a higher standard than that considered here, where the relevant comparison is with current *broadcast* standards (NTSC/PAL/SECAM).

0098-4094/87/1100-1385$01.00