A Flexible Parallel Architecture for Relaxation Labeling Algorithms

Shaw-Yin Lin and Zen Chen

Abstract—**The design of a flexible parallel architecture for both the discrete relaxation labeling (DRL) algorithm and the probabilistic relaxation labeling (PRL) algorithm is addressed. Through the analysis of parallelism in the computational models of both algorithms, the parallel execution of the algorithms on a flexible parallel architecture is presented. Three basic types of parallel operations are performed in the architecture: simultaneous, pipeline, and systolic. An illustrative example is used to show how the DRL algorithm can be executed on the parallel architecture. In doing so the processing element (PE) organization and the combiner organization of the architecture are described. The same architecture with programmable functional units is shown to be able to execute the PRL algorithm, too. The performance comparisons between the proposed architecture and some other existing ones are also given.**

I. Introduction

RELAXATION labeling algorithms have been applied successfully to a number of applications, for instance, signal restoration, language identification, graph homomorphism, image segmentation, and scene interpretation [1]-[8]. One of the major issues of using this type of algorithms is that it is computationally intensive and requires typically an exponential execution time on a single processor architecture [9]. Many researchers have tried to build fast architectures [9]-[15] for executing the relaxation labeling algorithms. Uresin and Dubois [10], Zenios and Mulvey [7], and Kamada *et al.* [11] proposed the multiprocessor approaches. They were faced with the difficulties in task partitioning, scheduling, and synchronization among multiprocessors. Resis and Kumar [12], and Derin and Won [13] used mesh connected computer (MCC) architectures to approach the relaxation problems. Their techniques have an appealing run time performance from a theoretical viewpoint, but suffer from physical realization problems such as data communication congestion and I/O routing complexities. Most of the above approaches remain in the phase of virtual software simulations [9]. Lately, there has been an increasing interest in implementing relaxation operations by using dedicated hardware architecture that can be built in a modular fashion, for instance, in the form of systolic arrays. Several preferential factors of the systolic approach, such as the feasibility of pipelining, high degree of parallelism, and avoidance of global communication, make the arrays suitable for VLSI fabrication. Among these architectures are the works by Derin and Won [13], Gu *et al.* [14], Guerra [15], and Bridges *et al.* [16]. The underlying idea behind these methods is the application of suitable transformations to the relaxation algorithms to obtain a representation that can be easily mapped onto their proposed architecture. Thus, different relaxation labeling algorithms give rise to different array designs. However, the function of resultant systolic array is somewhat restricted simply because the applied architecture is too fixed to cover different applications.

Manuscript received August 28, 1989; revised April 17, 1991.

The authors are with the Department of Computer Science and Information Engineering, National Chiao Tung University, Hsinchu, Taiwan, 30050, Republic of China.

IEEE Log Number 9106553.

In this paper, we consider the relaxation labeling algorithms in two different forms: discrete relaxation labeling (DRL) and probabilistic relaxation labeling (PRL). Both of them are executable on our proposed architecture. The arrays of the architecture use one-dimensional, one-way communication lines between adjacent PE's and interact with the external environment through a single I/O port. Because of the hardware simplicity and programmability features of the PE's, the architecture is well suited for VLSI implementation and is flexible enough to be adaptable to a number of applications. Moreover, the proposed architecture will run in a linear time for each iteration of labeling process. It is also able to check the consistency status of the DRL algorithm and the convergence condition of the PRL algorithm at the hardware level without the host involvement. On the other hand, the use of one-way flow through the arrays simplifies the circuit design and the system can be converted to self-timed arrays to avoid the clock skew problem for large labeling processes [18].

The paper consists of five sections. Section II presents the overview of the two relaxation labeling algorithms: DRL and PRL. In Section III, parallelism in the two relaxation mathematical models is the key to our design of a flexible parallel architecture. Three basic types of parallel operations are used: simultaneous, pipeline, and systolic. The processing element organization of the systolic array and the combiner organization are then introduced. An illustrative example is used to show how the DRL algorithm is executed on the architecture. The same architecture with programmable functional units is also shown to be able to execute the PRL algorithm. Section IV covers the performance comparisons between our architecture and some other existing ones. The final section is a summary.

II. Overview of Two Relaxation Labeling Algorithms

Given a set of N objects $U = \{U_1, U_2, \cdots, U_N\}$ and a set of M labels $\Lambda = \{\lambda_1, \lambda_2, \cdots, \lambda_M\}$, a relaxation labeling algorithm attempts to assign iteratively the labels to all objects such that these object-label assignments are consistent with a set of prespecified compatibility constraints (or coefficients) between the pairs of object-label assignments. We shall consider two types of algorithms: discrete relaxation labeling (DRL) and probabilistic relaxation labeling (PRL).

A. Discrete Relaxation Labeling (DRL)

In this case the labels are assigned to objects in an all-or-none fashion. The prespecified compatibility coefficients $CC = \{C_{i,j}(\lambda_t, \lambda_p), i, j \in [1, N]; t, p \in [1, M]\}$ are such that $C_{i,j}(\lambda_t, \lambda_p) = 1$ if the assignment of label λ_t to objects U_i is compatible with the assignment of label λ_p to object U_j and 0 otherwise. In the iterative relaxation labeling process for each object U_i, $i \in [1, N]$, let $L_i^k(\Lambda) = [L_i^k(\lambda_1), L_i^k(\lambda_2), \cdots, L_i^k(\lambda_M)]$ denote the vector of the label assignments given to object U_i at the kth iteration, $k = 1, 2, 3, \cdots$, where $L_i^k(\lambda_t) = 1$ if label λ_t is assigned to U_i and 0 otherwise. The complete list of object-label assignments, called a labeling $\bar{L}^k$, is indicated by $\bar{L}^k = \{L_1^k(\Lambda), L_2^k(\Lambda), \cdots, L_N^k(\Lambda)\}$. Initially, each object U_i is assigned to have all labels in Λ, i.e., $L_i^k(\lambda_t) = 1$ for $k = 0$, $i \in [1, N]$ and $t \in [1, M]$. Obviously, these initial assignments may not be consistent with the prespecified compatibility coefficients, so the labeling will be updated through the following formula:

$$L_i^{k+1}(\lambda_t) = L_i^k(\lambda_t) * \prod_{j=1}^{N} \left[\sum_{p=1}^{M} C_{i,j}(\lambda_t, \lambda_p) * L_j^k(\lambda_p) \right]$$
$$k = 0, 1, 2, 3, \cdots \qquad (1)$$

where $*$, Π, and Σ denote the Boolean AND, PRODUCT, and SUM operations, respectively. When there exists some finite constant K such that $k \geq K$, $L_i^{k+1}(\lambda_t) = L_i^k(\lambda_t)$ for all $i \in [1, N]$ and $t \in [1, M]$, then the labeling process is said to satisfy the consistency condition and the process stops.

B. Probabilistic Relaxation Labeling (PRL)

The PRL algorithm can be thought as a generalization of the DRL algorithm. The algorithm assigns different probabilities, instead of the zero-or-one fashion, to the object-label pairs. The probabilistic labeling estimates of object $U_i \in U$, denoted by $P_i(\lambda_t)$, $t \in [1, M]$, are ranged in the interval [0, 1] and will be updated during each iteration. The heuristic knowledge embedded in what are termed compatibility coefficients $CC = \{C_{i,j}(\lambda_t, \lambda_p), i, j \in [1, N]; t, p \in [1, M]\}$ is to control the contribution that the probability of assigning label λ_p to object U_j made to the probability of assigning label λ_t to object U_i. Historically, $C_{i,j}(\lambda_t, \lambda_p)$ is so defined that it takes a value in the range of $[-1, 1]$ where -1, 0 and 1 indicate "totally incompatible," "independent," and "totally compatible," respectively. The initial labeling estimate of $P_i^k(\lambda_t)$, for $k = 0$, $i \in [1, N]$ and $t \in [1, M]$, is estimated based on the information available on hand. The iterative updating of the labeling of object U_i, $i \in [1, N]$, is given by

$$P_i^{k+1}(\lambda_t) = \frac{P_i^k(\lambda_t)\,[1 + S_i^k(\lambda_t)]}{\sum_{p=1}^{M} P_i^k(\lambda_p)\,[1 + S_i^k(\lambda_p)]} \qquad (2)$$

for $i \in [1, N]$, $t \in [1, M]$, and $k = 0, 1, 2, 3, \cdots$, where

$$S_i^k(\lambda_t) = \sum_{j=1}^{N} \sum_{p=1}^{M} C_{i,j}(\lambda_t, \lambda_p)\, P_j^k(\lambda_p).$$

The updating of these labeling estimates stops if the estimates are unchanged or nearly unchanged after a certain finite number of iterations. In this case, it is said a convergence condition is reached.

III. Parallel Execution of Relaxation Labeling Algorithms

A. The DRL Algorithm

In the following we shall examine the parallelism in the computation models of both DRL and PRL algorithms. From this analysis of parallelism we shall determine three basic types of parallel operations for hardware execution of these algorithms. We shall first describe the architecture for the DRL algorithm, then point out the necessary modifications of the DRL architecture needed in order to execute the PRL algorithm.

To illustrate the idea more explicitly, let us consider a region color labeing problem [13]. Suppose that we are analyzing a picture with five regions which are to be colored in red, green, and blue, subject to certain compatibility constraints. The DRL formulation for this problem is given below:

1) A set of five regions $U = \{U_1, U_2, U_3, U_4, U_5\}$ where U_i = region i. The object set size is $N = 5$.

2) The coloring labels $\Lambda = \{\lambda_1, \lambda_2, \lambda_3\}$ where λ_1 is red, λ_2 is green, and λ_3 is blue. The label set size is $M = 3$.

3) The compatibility coefficients (or constraints) between any two region label assignments $C_{i,j}(\lambda_t, \lambda_p)$ $i, j \in [1, 5]$ and $t, p \in [1, 3]$.

Let us rewrite (1) as follows:
For $i, j \in [1, 5]$ and $t, p \in [1, 3]$

$$S_{i,j}^k(\lambda_t) = \sum_{p=1}^{3} C_{i,j}(\lambda_t, \lambda_p) * L_j^k(\lambda_p) \qquad (3a)$$

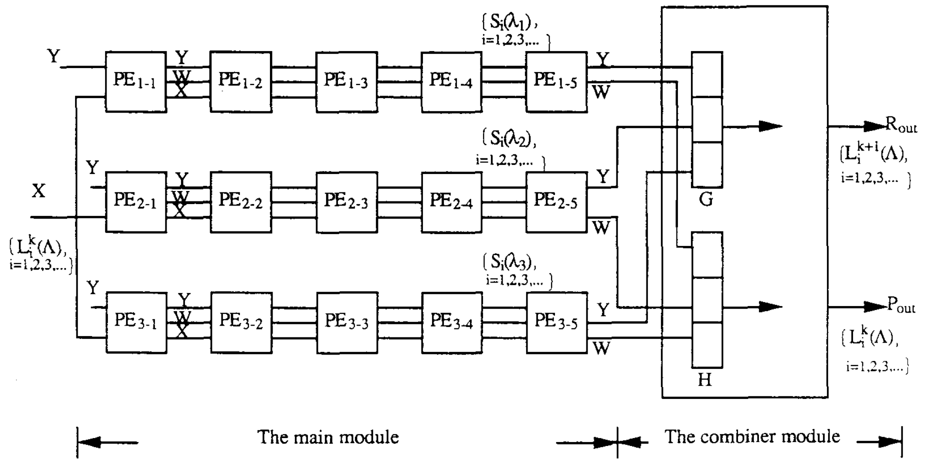

Y
X
{L$_i^k$(Λ), i=1,2,3,...}
PE1-1
PE1-2
PE1-3
PE1-4
PE1-5
PE2-1
PE2-2
PE2-3
PE2-4
PE2-5
PE3-1
PE3-2
PE3-3
PE3-4
PE3-5
Y
W
X
{S$_i$(λ$_1$), i=1,2,3,... }
{S$_i$(λ$_2$), i=1,2,3,... }
{S$_i$(λ$_3$), i=1,2,3,... }
G
H
R$_{out}$
{L$_i^{k+1}$(Λ), i=1,2,3,... }
P$_{out}$
{L$_i^k$(Λ), i=1,2,3,...}
The main module
The combiner module

Fig. 1. The organization of the proposed architecture.

TABLE I
THE LIST OF DATA STREAMS DURING EACH CLOCK CYCLE FOR THE DRL EXAMPLE: THE INPUT DATA STREAMS AT THE Y_{in} AND X_{in} ENDS

Clock	0	1	2	3	4	5	6	7	8	9	10	11	⋯	25	26	27
Y_{in}	1	1	1	1	1	1	1	1	1	1	1	1	⋯	1	1	1
X_{in}	$L_1^k(\Lambda)$	$L_2^k(\Lambda)$	$L_3^k(\Lambda)$	$L_4^k(\Lambda)$	$L_5^k(\Lambda)$	$L_1^k(\Lambda)$	$L_2^k(\Lambda)$	$L_3^k(\Lambda)$	$L_4^k(\Lambda)$	$L_5^k(\Lambda)$	0	0	⋯	0	$L_1^{k+1}(\Lambda)$	$L_2^{k+1}(\Lambda)$

$$S_i^k(\lambda_t) = \prod_{j=1}^{5} S_{i,j}^k(\lambda_t) \tag{3b}$$

$$L_i^{k+1}(\lambda_t) = L_i^k(\lambda_t) * S_i^k(\lambda_t). \tag{3c}$$

All computations in (3) will be executed in parallel on a parallel architecture to be introduced below. There are totally $N \times M = 15$ supporting evidences, i.e., $S_i(\lambda_t)$, $i = 1, 2, 3, 4, 5$ and $t = 1, 2, 3$. They are divided into three groups: $\{S_1(\lambda_1), S_2(\lambda_1), S_3(\lambda_1), S_4(\lambda_1), S_5(\lambda_1)\}$, $\{S_1(\lambda_2), S_2(\lambda_2), S_3(\lambda_2), S_4(\lambda_2), S_5(\lambda_2)\}$, and $\{S_1(\lambda_3)$ $S_2(\lambda_3), S_3(\lambda_3), S_4(\lambda_3), S_5(\lambda_3)\}$. These three groups are to be executed, respectively, by three linear rows of processing elements (PE's). The physical arrangement of these computations is shown in Fig. 1. All computations are done in three different ways:

1) Simultaneous computation. The computations of $S_i^k(\lambda_1)$, $S_i^k(\lambda_2)$, and $S_i^k(\lambda_3)$, $i \in [1, 5]$, specified by (3b) are simultaneously executed in the three rows of PE's.

2) Pipeline computation. The computations of $S_1(\lambda_t)$, $S_2(\lambda_t)$, $S_3(\lambda_t)$, $S_4(\lambda_t)$, and $S_5(\lambda_t)$, are computed in a pipeline order in the tth row where $t \in [1, 3]$. Also the new labeling estimates $L_1^{k+1}(\lambda_t)$, $L_2^{k+1}(\lambda_t)$, $L_3^{k+1}(\lambda_t)$, $L_4^{k+1}(\lambda_t)$, and $L_5^{k+1}(\lambda_t)$, specified by (3c) are generated in a pipeline order in the combiner module, too.

3) Systolic computation. The Boolean product specified by (3b) for some $i \in [1, 5]$ is executed in a systolic manner, because it involves the input stream of labeling estimates $L_j^k(\cdot), j = 1, 2, \cdots, 5$, in (3a). The detailed description will be given later. One the other hand, since all variables $S_{i,j}^k(\lambda_t)$, $C_{i,j}(\lambda_t, \lambda_p)$ and $L_j^k(\lambda_p)$ in (3a) are 1-b wide, we can use the logic gates with 1-b inputs to implement (3a). However, we shall pack M 1-b labels λ_1, $\lambda_2, \cdots, \lambda_M (M = 3)$ into an M-bit label vector Λ to represent all labels. So (3a) will be executed by a Boolean circuit with M-bit inputs. From now on, we use $L_i^k(\Lambda)$ to represent the 3-b vector $[L_i^k(\lambda_1), L_i^k(\lambda_2), L_i^k(\lambda_3)]$.

Next we shall describe, in more detail, i) how the systolic computation of $S_1^k(\lambda_1)$ is done in the first row of five PE's in the main module, and ii) how the pipeline computation of $L_1^{k+1}(\Lambda)$ is done in the combiner module. We shall introduce the required hardware architecture, too.

First of all, the input data streams at the Y_{in} and X_{in} ends of the first PE's of all the three rows are given in Table I. These input streams will step through the PE's in each row simultaneously. The three pipeline stages of a PE, as shown in Fig. 2(a), are so designed that:

1) The first PFIFO stage is to delay the input stream at X_{in} end of the first PE for $N - 1 = 4$ clocks in order to synchronize the $L_i^k(\Lambda)$ stream with the $S_i^k(\Lambda)$ stream generated at the output end of the last PE for computing the new labeling estimate $L_i^{k+1}(\Lambda)$ stream specified in (3c). The delayed $L_i^k(\Lambda)$ stream is fed to the W line.

2) The I stage is to compute $S_{1,j}^k(\lambda_1)$ of (3a) for some $j \in [1, 5]$.

3) The A stage is to compute some partial product of $S_1^k(\lambda_1)$ of 3(b).

4) The Z buffer is to adjust the data flow rate on the X_{out} line such that it is one clock behind that on the Y line. This synchronization is needed in the systolic operation for executing (3b).

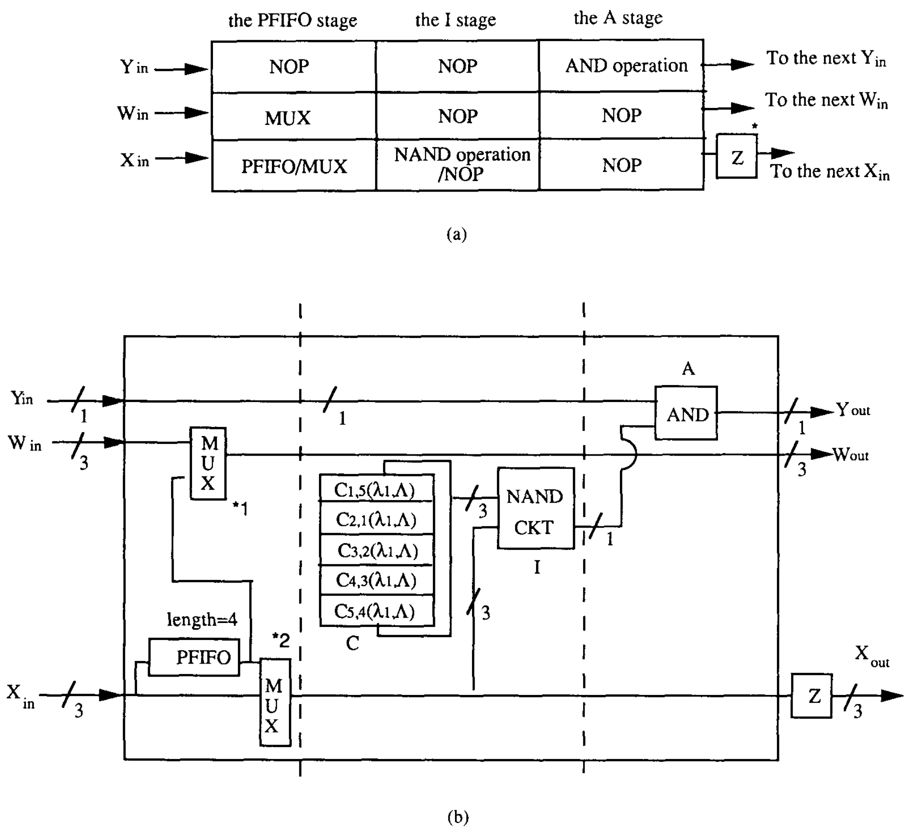


Fig. 2. (a) The three pipeline stages of a PE. *Z is a one-clock delay buffer. (b) The hardware organization of the PE for the implementation of the DRL algorithm. *1: The multiplexer selects the input from PFIFO in the first PE of each row, and it selects the W_{in} input in all the subsequent PE's. *2: For the DRL mode, this multiplexer always selects the bottom input.

Let us see how the actual systolic operation takes place in the first row of PE's. For instance,

$$\begin{aligned} S^k_1(\lambda_1) &= \prod_{j=1}^{5} S^k_{1,j}(\lambda_1) \\ &= \underbrace{\underbrace{\underbrace{\underbrace{\underbrace{[[[[S^k_{1,5}(\lambda_1)]}_{PE_{1-1}} * S^k_{1,4}(\lambda_1)]}_{PE_{1-2}} * S^k_{1,3}(\lambda_1)]}_{PE_{1-3}} * S^k_{1,2}(\lambda_1)]}_{PE_{1-4}} * S^k_{1,1}(\lambda_1)]}_{PE_{1-5}}. \end{aligned}$$

All the partial Boolean products of $S^k_1(\lambda_1)$ will be generated one at a PE in a fixed order from PE_{1-1} to PE_{1-5}, as shown above. The first partial product is $S^k_{1,5}(\lambda_1)$ in which we need $L^k_5(\Lambda)$. $L^k_5(\Lambda)$ arrives at the X_{in} end of PE_{1-1} during clock 4 and passes through the PFIFO stage at clock 5. The computation of $S^k_{1,5}(\lambda_1) = \Sigma^3_{p=1} C_{1,5}(\lambda_1, \lambda_p) * L^k_5(\lambda_p)$ is performed at the I stage by a two-level NAND circuit during clock 6. In the cycle of clock 7, the first partial product $\Pi^5_{j=5} S^k_{1,j}(\lambda_1)$ is computed at the A stage of PE_{1-1} by an AND gate. The result is output through the Y_{out} line. This time-space diagram is shown in

TABLE II
THE TIME-SPACE SNAPSHOTS OF PE CONTENTS IN EACH CLOCK CYCLE FOR COMPUTING THE PARTIAL PRODUCTS OF $S_i^k(\lambda_1)$ IN PE_{1-1} AND PE_{1-2}

	Clock	3	4	5	6	7	8	9	10	$\cdots$
	X_{in}	$L_4^k(\Lambda)$	$L_5^k(\Lambda)$	$L_1^k(\Lambda)$	$L_2^k(\Lambda)$	$L_3^k(\Lambda)$	$L_4^k(\Lambda)$	$L_5^k(\Lambda)$	0	
	FIFO Stage		$L_4^k(\Lambda)$	$L_5^k(\Lambda)$	$L_1^k(\Lambda)$	$L_2^k(\Lambda)$	$L_3^k(\Lambda)$	$L_4^k(\Lambda)$	$L_5^k(\Lambda)$	
PE_{1-1}	I Stage				$S_{1,5}^k(\lambda_1)$					
	A Stage					$1 * S_{1,5}^k(\lambda_1)$				
	FIFO Stage		0	$L_1^k(\Lambda)$	$L_2^k(\Lambda)$	$L_3^k(\Lambda)$	$L_4^k(\Lambda)$	$L_5^k(\Lambda)$		
PE_{1-2}	I Stage							$S_{1,4}^k(\lambda_1)$		
	A Stage								$\Pi_{j=4}^5 S_{1,j}^k(\lambda_1)$	

TABLE III
THE LIST OF DATA STREAMS DURING EACH CLOCK CYCLE FOR THE DRL EXAMPLE: THE GENERATED $S_i^k(\lambda_1)$, $S_i^k(\lambda_2)$, $S_i^k(\lambda_3)$, AT THE Y_{out} ENDS OF THREE LINEAR ARRAYS

Clock	19	20	21	22	23	24	25	$\cdots$	44	45	46
Row 1	$S_1^k(\lambda_1)$	$S_2^k(\lambda_1)$	$S_3^k(\lambda_1)$	$S_4^k(\lambda_1)$	$S_5^k(\lambda_1)$	0	0	$\cdots$	0	$S_1^{k+1}(\lambda_1)$	$S_2^{k+1}(\lambda_1)$
Row 2	$S_1^k(\lambda_2)$	$S_2^k(\lambda_2)$	$S_2^k(\lambda_2)$	$S_4^k(\lambda_2)$	$S_5^k(\lambda_2)$	0	0	$\cdots$	0	$S_1^{k+1}(\lambda_2)$	$S_2^{k+1}(\lambda_2)$
Row 3	$S_1^k(\lambda_3)$	$S_2^k(\lambda_3)$	$S_3^k(\lambda_3)$	$S_4^k(\lambda_3)$	$S_5^k(\lambda_3)$	0	0	$\cdots$	0	$S_1^{k+1}(\lambda_3)$	$S_2^{k+1}(\lambda_3)$

Table II. Similarly, $S_{1,4}^k(\lambda_1) = \Sigma_{p=1}^3 C_{1,4}(\lambda_1, \lambda_p) * L_4^k(\lambda_p)$ is computed at the I stage of PE_{1-2} during clock 9. The partial product $\Pi_{j=4}^5 S_{1,j}^k(\lambda_1)$ is then generated at the A stage during clock 10. The growth of the partial product continues to take place at PE_{1-3}, PE_{1-4}, and PE_{1-5}. And the final result of $S_1^k(\lambda_1) = \Pi_{j=1}^5 S_{1,j}^k(\lambda_1)$ is obtained during clock 19, as indicated in Table III.

$S_1^k(\lambda_2)$ and $S_1^k(\lambda_3)$ are simultaneously generated in the second and third rows, when $S_1^k(\lambda_1)$ is generated in the first row. The subsequent supporting evidences $S_i^k(\lambda_t)$ for $i = 2, 3, \cdots$, are generated in the same way during the clock cycles given in Table III.

The correct values of supporting evidence generated above in each row rely on the proper pairing between the compatibility coefficients and the labeling estimates, as given in (3a). The compatibility coefficients are arranged in a circular shift register in the five PE's of the first row and are ordered according to the required timing relations needed to compute $S_1(\lambda_1)$, $S_2(\lambda_1)$, $S_3(\lambda_1)$, $S_4(\lambda_1)$, and $S_5(\lambda_1)$ in the first row of Fig. 3.

The compatibility coefficient arrangements in the PE's of other rows in Fig. 3 can be obtained by simply changing λ_1 to λ_2 for the second row, and changing λ_1 to λ_3 for the third row.

Next, consider (3c) for computing the new labeling estimates. The generated new labeling estimate stream at the R_{out} end is shown in Table IV. Let us see how $L_1^{k+1}(\Lambda)$ is generated during clock 24 in the combiner module. The combiner organiztion consists of five pipeline stages, as shown in Fig. 4. It takes five clocks to generate a new labeling estimate. At clock 20, the generated $S_1^k(\lambda_1)$, $S_1^k(\lambda_2)$, and $S_1^k(\lambda_3)$, are output as $S_1^k(\Lambda)$, by a parallel-in-and-parallel-output (PIPO) G registers at the PIPO stage. The labeling estimate $L_1^k(\Lambda)$ passes through the PIPO stage via H registers at clock 20, too. At clock 21, $S_1^k(\Lambda)$ passes through the A stage without any operation. At clock 22, the Boolean ANDing operation of $L_1^k(\Lambda) * S_1^k(\Lambda) = L_1^{k+1}(\Lambda)$ is performed at the I stage. At clocks 23 and 24, the result $L_1^{k+1}(\Lambda)$ steps through the last two stages without any operations. Thus at the end of clock 24, $L_1^{k+1}(\Lambda)$ appears at the R_{out} end. The reason for using the NOP (no operation) mode of A, ACC, D stages in the combiner is to allow this same combiner structure to be used in a programmable mode to handle the probabilistic relaxation labeling (PRL) algorithm. The other new labeling estimates are generated similarly in a pipeline order, as shown in Table IV. At clock 25 the new stream of $L_i^{k+1}(\Lambda)$, $i = 1, 2, \cdots$, and the old stream of $L_i^k(\Lambda)$, $i = 1, 2, \cdots$, obtained at the R_{out} and P_{out} ends are compared at the hardware level to determine the consistency condition. At clock 26 the new stream of $L_i^{k+1}(\Lambda)$, $i = 1, 2, \cdots$, starts to appear at X_{in} end.

This completes our description of the parallel execution (or mapping) of the given DRL algorithm on our parallel

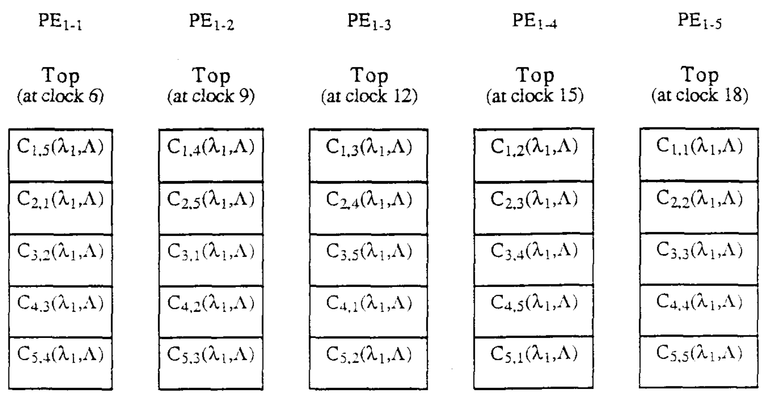


Fig. 3. The compatibility coefficient arrangements in the PE's of the first row.

TABLE IV
THE LIST OF DATA STREAMS DURING EACH CLOCK CYCLE FOR THE DRL EXAMPLE: THE NEW AND OLD LABELING ESTIMATE STREAMS AT R_{out} AND P_{out} ENDS

Clock	24	25	26	27	28	29	30	···	49	50
R_{out}	$L_1^{k+1}(\Lambda)$	$L_2^{k+1}(\Lambda)$	$L_3^{k+1}(\Lambda)$	$L_4^{k+1}(\Lambda)$	$L_5^{k+1}(\Lambda)$	0	0	···	0	$L_1^{k+2}(\Lambda)$
P_{out}	$L_1^{k}(\Lambda)$	$L_2^{k}(\Lambda)$	$L_3^{k}(\Lambda)$	$L_4^{k}(\Lambda)$	$L_5^{k}(\Lambda)$	0	0	···	0	$L_1^{k+1}(\Lambda)$

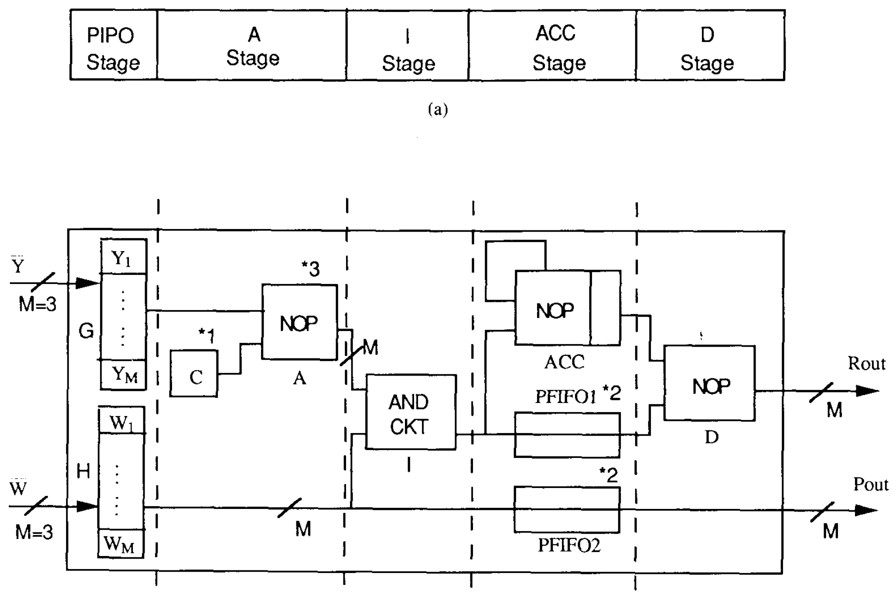
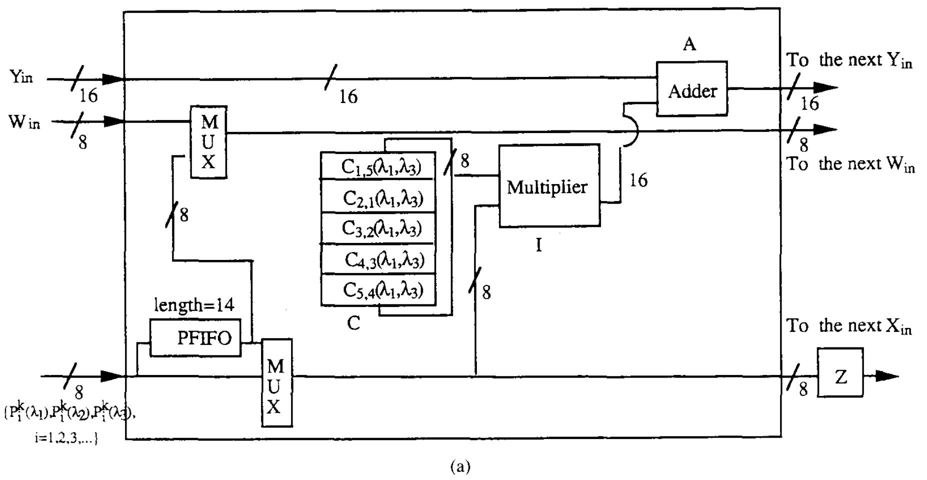
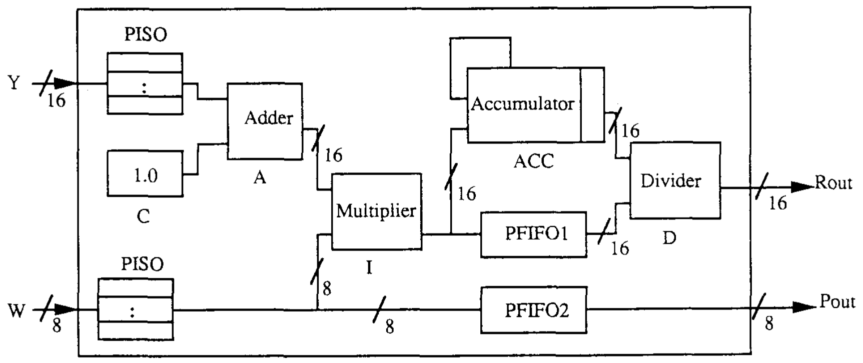


Fig. 4. (a) The five pipeline stages of the combiner module. (b) The hardware organization of the combiner module. *1: The constant buffer is not used in the DRL mode. *2: Both PFIFO1 and PFIFO2 are short circuited in the DRL mode. *3: The A, ACC, and D stages are set to the no operation mode; the input data pass through intact.



(a)



(b)

Fig. 5. (a) The hardware organization of PE and (b) the hardware organization of combiner module for the implementation of the PRL algorithm.

architecture. Next, we shall modify this parallel architecture so that it can execute the PRL algorithm, too.

B. The PRL Algorithm

Let us rewrite a 5-object, 3-label PRL algorithm as follows:

$$S_{i,j}^k(\lambda_t) = \sum_{p=1}^{3} C_{i,j}(\lambda_t, \lambda_p)\, P_j^k(\lambda_p) \tag{4a}$$

$$S_i^k(\lambda_t) = \sum_{j=1}^{5} S_{i,j}^k(\lambda_t) \tag{4b}$$

$$P_i^{k+1}(\lambda_t) = \frac{P_i^k(\lambda_t)\,[1 + S_i^k(\lambda_t)]}{\sum_{p=1}^{3} P_i^k(\lambda_p)\,[1 + S_i^k(\lambda_p)]}$$

$$\text{for } i, j \in [1, 5] \quad \text{and} \quad p \in [1, 3]. \tag{4c}$$

In (4a) $S_{i,j}^k(\lambda_t)$, $C_{i,j}(\lambda_t, \lambda_p)$, and $P_j^k(\lambda_p)$ are real numbers instead of 1-b-wide binary numbers as in the DRL case. A real number here is represented by 8 b. We shall not pack M 8-b labeling estimates $P_j^k(\lambda_1)$, $P_j^k(\lambda_2)$, $P_j^k(\lambda_3)$ into one 3×8-bit datum as before, because it is not practical to use such a 24-b-wide data at the input line. Instead, we will use the 8-b-wide data to represent our input stream. We shall implement $S_{i,j}^k(\lambda_t)$ in (4a) by a linear systolic array of three PE's. On the other hand, $S_i^k(\lambda_t)$ in (4b) is again computed through a systolic operation as in the DRL case. In total, we need $3 \times 5 = 15$ PE's in order to obtain the final result of $S_i^k(\lambda_t)$ for each $i \in [1, 5]$.

The necessary modifications of the previous DRL architecture for executing the PRL algorithm are as follows. The architecture modifications are shown in Fig. 5.

1) The I stage is configured as a multiplier.
2) The A stage is configured as an adder.
3) The number of PE's in each row is extended to 15.
4) The lengths of the PFIFO buffer in the first PE's of the three rows are 14, 13, and 12, respectively, where $14 = N \times M - 1$.

TABLE V
THE LIST OF DATA STREAMS DURING EACH CLOCK CYCLE FOR THE PRL EXAMPLE: THE INPUT DATA STREAMS AT THE Y_{in} AND X_{in} ENDS

Clock	0	1	2	3	$\cdots$	14	15	16	$\cdots$
Y_{in}	0	0	0	0	$\cdots$	0	0	0	$\cdots$
X_{in}	$P_1^k(\lambda_1)$	$P_1^k(\lambda_2)$	$P_1^k(\lambda_3)$	$P_2^k(\lambda_1)$	$\cdots$	$P_5^k(\lambda_3)$	$P_1^k(\lambda_1)$	$P_1^k(\lambda_2)$	$\cdots$

TABLE VI
THE LIST OF DATA STREAMS DURING EACH CLOCK CYCLE FOR THE PRL EXAMPLE: THE GENERATED $S_i^k(\lambda_1)$, $S_i^k(\lambda_2)$, $S_i^k(\lambda_3)$ AT THE Y_{out} ENDS OF THREE LINEAR ARRAYS

Clock	59	60	61	62	$\cdots$	71	$\cdots$
Row 1	$S_1^k(\lambda_1)$	—	—	$S_2^k(\lambda_1)$	$\cdots$	$S_5^k(\lambda_1)$	$\cdots$
Row 2	$S_1^k(\lambda_2)$	—	—	$S_2^k(\lambda_2)$	$\cdots$	$S_5^k(\lambda_2)$	$\cdots$
Row 3	$S_1^k(\lambda_3)$	—	—	$S_2^k(\lambda_3)$	$\cdots$	$S_5^k(\lambda_3)$	$\cdots$

TABLE VII
THE LIST OF DATA STREAMS DURING EACH CLOCK CYCLE FOR THE PRL EXAMPLE: THE NEW AND OLD LABELING ESTIMATE STREAMS AT R_{out} AND P_{out} ENDS

Clock	66	67	68	69	$\cdots$	80	$\cdots$
R_{out}	$P_1^{k+1}(\lambda_1)$	$P_1^{k+1}(\lambda_2)$	$P_1^{k+1}(\lambda_3)$	$P_2^{k+1}(\lambda_1)$	$\cdots$	$P_5^{k+1}(\lambda_3)$	$\cdots$
P_{out}	$P_1^k(\lambda_1)$	$P_1^k(\lambda_2)$	$P_1^k(\lambda_3)$	$P_2^k(\lambda_1)$	$\cdots$	$P_5^k(\lambda_3)$	$\cdots$

TABLE VIII
THE DIFFERENT MODES OF THE FUNCTIONAL UNITS FOR THE EXECUTION OF THE DRL AND PRL ALGORITHMS

Units / Cases	PE		The Combiner Module				
	I	A	G H	A	I	ACC	D
DRL	NAND ckt	AND	PIPO	NOP	AND	NOP	NOP
PRL	Multiplier	Adder	PISO	Adder	Multiplier	Accumulator	Divider

The computation of a new labeling estimate $P_i^{k+1}(\lambda_t)$ according to (4c) using the combiner is more complicated than before. The modifications include

1) The first pipeline stage is changed to parallel-in-and-serial-out (PISO) registers.
2) The constant buffer C is set to 1.0 to get the term of $1 + S_i^k(\lambda_t)$.
3) The A stage is configured as an adder.
4) The I stage is configured as a multiplier.
5) The ACC stage is configured as an accumulator with a buffer length of $M = 3$.
6) The lengths of PFIFO1 and PFIFO2 are set to $M = 3$.
7) The D stage is configured as a divider.

Tables V–VII show the data streams at the clock steps for the generation of new estimates $P_1^{k+1}(\lambda_1)$, $P_1^{k+1}(\lambda_2)$, $\cdots$. Let us take a look at a snapshot of clock steps for $P_1^{k+1}(\lambda_1)$ generated in the combiner. During clock 59, where $59 = 3 \times N \times M + N \times M - 1$, the result of $S_1^k(\lambda_1)$ is available at Y_{out} end and the $P_1^k(\lambda_1)$ is available at W_{out} end. During the next four clock cycles, the terms $[1 + S_1^k(\lambda_1)]$, $[1 + S_1^k(\lambda_1)] \times P_1^k(\lambda_1)$ and the partial sum of 0 and $[1 + S_1^k(\lambda_1)] \times P_1^k(\lambda_1)$ are obtained with adder, multiplier, and accumulator units. Similarly, in the clock cycles 62, 63, and 64, the terms $[1 + S_1^k(\lambda_2)]$, $[1 + S_1^k(\lambda_2)] \times P_1^k(\lambda_2)$ and partial sum of $[1 + S_1^k(\lambda_1)] \times P_1^k(\lambda_1)$ and $[1 + S_1^k(\lambda_2)] \times P_1^k(\lambda_2)$ are obtained, and then the $\Sigma_{p=1}^3 [1 + S_1^k(\lambda_p)] \times P_1^k(\lambda_p)$ is obtained in clock 65. Finally, the new labeling estimate $P_1^{k+1}(\lambda_1)$ is produced by the divider and is output at the R_{out} end in clock 66. The above process operates in a pipeline fashion, so the succeeding new estimates of $P_1^{k+1}(\lambda_2)$, $P_1^{k+1}(\lambda_3)$, $P_2^{k+1}(\lambda_1)$, $\cdots$ will be produced consecutively on R_{out} line at the rate of one per clock cycle. Totally, it takes 80 clock cycles to finish one iteration of the PRL agorithm.

C. The General Parallel Architecture for DRL and PRL Algorithms

From above, we can see a common parallel architecture can be used for executing DRL and PRL algorithms. The differences in the PE and combiner hardware organizations for these two cases can be settled by using programmable units in the organizations. The different modes of these functional units for the DRL and PRL cases are shown in Table VIII.

TABLE IX
COMPARISONS OF PROPOSED ARCHITECTURE WITH TWO OTHER ARCHITECTURES FOR THE DRL ALGORITHMS

No.	Item of Comparison	Architecture by Gu *et al.*	Architecture by Resis *et al.*	Our Architecture
1)	Type of architecture	2-D, 2-W† systolic array	2-D mesh connected computer	1-D, 1-W systolic array
2)	Time complexity per iteration	$O(N \times M)$†	$O(N \times M^2)$	$O(N)$
3)	Space complexity	$O(N \times M)$	$O(N^2)$	$O(N \times M)$
4)	I/O bandwidth in bits	$O(M \times M)$	$O(N \times M)$	$O(M)$
5)	Configurability	No	No	Yes
6)	Adaptibility to varying problem sizes	limited	No	Yes
7)	Consistency checking	Yes	No	Yes

†2-D, 2-W = two-dimensional, two-way; M = the number of labels; N = the number of objects.

TABLE X
COMPARISONS OF PROPOSED ARCHITECTURE WITH TWO OTHER ARCHITECTURES FOR THE PRL ALGORITHMS

No.	Item of Comparison	Architecture by Kamada *et al.*	Architecture by Guerra *et al.*	Our Architecture
1)	Type of architecture	Round robin multiprocessor	1-D, 1-W† systolic array	1-D, 1-W systolic array
2)	Time complexity	$O(N \times M)$†	$O(N^2 \times M)$	$O(N \times M)$
3)	Space complexity	$O(N)$	$O(N \times M)$	$O(N \times M^2)$
4)	I/O channel number	$O(M)$	$O(M)$	1
5)	Configurability	No	No	Yes
6)	Adaptibility to varying problem sizes	difficult	Yes	Yes
7)	Convergence status checking	No	No	Yes

†1-D, 1-W = one-dimensional, one-way; M = the number of labels; N = the number of objects.

IV. PERFORMANCE EVALUATION

To evaluate the proposed architecture, we shall consider several evaluation factors including the time complexity, space utilization, and input channel bandwidth. The comparisons of our architecture with other relevant architectures for both DRL and PRL cases are shown in Tables IX and X, respectively.

A. The DRL Algorithm

Table IX summarizes the differences between our architecture and those by Gu *et al.* [14] and Resis and Kumar [12] for the execution of the DRL algorithm. Assuming the clock cycle time is T, the time complexity per iteration of our architecture is estimated as follows:

1) The time to produce the first new labeling estimate. It consists of two parts: a) $(N - 1)T$ which is the time required to wait for the arrival of $L_N^k(\Lambda)$ in order to compute the first partial result of a supporting evidence, and b) $(3 \times N + 5)T$ which is the time to get through both the main module and the combiner module.

2) The time for computing the subsequent labeling estimates. There are $N - 1$ subsequent labeling estimates to be generated at the rate of one per clock cycle, so it takes $(N - 1)T$ to complete.

As a result, the time complexity for a single iteration is $O(N)$. This indicates that the computation time depends only on the number of objects N, not on the size of classes M, while the other two architectures require $O(NXM)$ and $O(NXM^2)$, respectively.

Table IX also lists several advantageous features of our architecture, such as the communication with the external environment through only one single I/O port, the simplicity of the architecture using one-dimensional, one-way systolic arrays and, the programmability of the functional units.

B. The PRL Algorithm

Table X summarizes the differences between our architecture and those by Kamada *et al.* [11] and Guerra [15] for executing the PRL algorithm. The time complexity per iteration is estimated as follows, assuming T is the clock cycle time:

1) The time to produce the first new labeling estimate. It consists of two parts: a) $(N \times M - 1)T$ which is the time required to wait for the arrival of $p_N^k(\lambda_M)$ in order to compute the first partial result of a supporting evidence, and b) $(3 \times N \times M + M + 4)T$ which is the time to get through the main module and the combiner module.

2) The time for computing the subsequent labeling estimates. There are $N \times M - 1$ subsequent new labeling

estimated to be generated at the rate of one per clock cycle, so it takes $(N \times M - 1)T$ to finish.

Therefore, the total time complexity for a single iteration is $O(N \times M)$. Although it is of the same order as the multiprocessor architecture proposed by Kamada *et al.*, yet the multiprocessor system takes a much longer clock cycle time than ours due to its complicated task scheduling and synchronization. On the other hand, ours is superior to the architecture proposed by Guerra as far as the time complexity is concerned. However, our architecture needs more PE's than the other two architectures. Nevertheless, our PE circuit is simpler and can be implemented in a high density VLSI chip. Furthermore, the convergence condition can be checked in our architecture at the hardware level without the host involvement, while the other two systems cannot do so.

V. Summary

We have presented a flexible parallel architecture for executing the DRL and PRL algorithms. The architecture is designed based on the analysis of the parallelism in the mathematical models of the algorithms. We preload the compatibility coefficients into the PE's so that only the stream of labeling estimates need to step through the PE's. The one-dimensional, one-way layout of systolic arrays is suitable for VLSI fabrication. The performance evaluations listed in Tables IX and X show that the proposed architecture is generally better than the existent architectures.

In order to verify the design of the proposed systolic array and the combiner module, a simulation software package called DAISY system [19] has been used to check the specification. We have finished the logic simulation to check the timing sequence for the architecture operation and the signal simulation to verify the intermediate processing results. The experiments show that the proposed architecture is working properly. Future work includes the development of the VLSI customer chip and proper architecture modifications to cover a wide spectrum of related algorithms.

References

[1] D. H. Ballard and C. M. Brown, *Computer Vision.* Englewood Cliffs, NJ: Prentice-Hall, 1982.

[2] A. Bundy, "Catalogue of artificial intelligence tools," in *Symbolic Computation Artificial Intelligence*, L. Bolc, A. Bundy, P. Hayes, and J. Siekmann, Eds. Berlin: Springer, 1984.

[3] B. Nudel, "Consistent-labeling problems and their algorithms: Expected-complexities and theory-based heuristics," *Artificial Intell.*, vol. 21, pp. 135-178, 1983.

[4] A. Goshtasby and R. W. Ehrich, "Contextual word recognition using probabilistic relaxation labeling," *Patt. Recog.*, vol. 21, no. 5, pp. 455-463, 1988.

[5] F. A. Mota and F. R. D. Velasco, "A method for the analysis of ambiguous segmentation of images," *IEEE Trans. Patt. Anal. Machine Intell.*, vol. PAMI-8, pp. 755-760, Nov. 1986.

[6] J. F. Boyce, J. Feng, and E. R. Haddow, "Relaxation labeling and the entropy of neighborhood information," *Pattern Recog. Lett.*, vol. 6, pp. 225-234, Sept. 1987.

[7] S. A. Zenois and J. M. Mulvey, "A distributed algorithm for convex network optimization problems," *Parallel Computing*, vol. 6, pp. 45-56, 1988.

[8] R. A. Hummel and S. W. Zucker, "On the foundations of relaxation labeling processes," *IEEE Trans. Patt. Anal. Machine Intell.*, vol. PAMI-5, pp. 267-287, May 1983.

[9] J. T. McCall, J. G. Tront, F. G. Gray, R. M. Haralic, and W. M. McCormack, "Parallel computer architecture and problem solving strategies for the consistent labeling problem," *IEEE Trans. Comput.*, vol. C-34, pp. 973-980, Nov. 1985.

[10] A. Uresin and M. Dubois, "Sufficient conditions for the convergence of asynchronous iteration," *Parallel Computing*, vol. 10, pp. 83-92, 1989.

[11] M. Kamada, K. Toraichi, R. Mori, K. Yamamoto, and H. Yamada, "A parallel architecture for relaxation operations," *Patt. Recog.*, vol. 21, pp. 175-181, 1988.

[12] D. Resis and V. K. P. Kumar, "Parallel processing of the labeling problem," in *Proc. IEEE Comput. Soc. Workshop Comput. Architecture Patt. Anal. Image Database Manag.*, 1985, pp. 381-385.

[13] H. Derin and C. S. Won, "A parallel image segmentation algorithm using relaxation with varying neighborhoods and its mapping to arrays processors," *Comput. Vision Graph., Image Processing*, vol. 40, pp. 54-78, 1987.

[14] J. Gu, W. Wang, and T. C. Henderson, "A parallel architecture for discrete relaxation algorithm," *IEEE Trans. Patt. Anal. Machine Intell.*, vol PAMI-9, pp. 816-830, Nov. 1987.

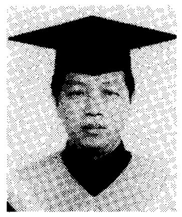
[15] C. Guerra, "Systolic algorithm for local operations on images," *IEEE Trans. Comput.*, vol. C-35, pp. 73-77, Jan. 1986.

[16] G. E. Bridges, W. Pries, R. D. McLeod, M. Yunik, P. G. Gulak, and H. C. Card, "Dual systolic architecture for VLSI digital signal processing systems," *IEEE Trans. Comput.*, vol. C-35, pp. 916-920, Oct. 1986.

[17] A. Rosenfeld, R. A. Hummel, and S. W. Zucker, "Scene labeling by relaxation operations," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-6, pp. 420-433, June 1976.

[18] O. H. Ibarra and H. A. Palis, "VLSI algorithms for solving recurrence equations and applications," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-35, pp. 1046-1052, July 1987.

[19] *Advanced Simulation*, Daisy System Corp., Student's Workbook, 1986.



Shaw-Yin Lin was born in Taiwan, Republic of China, in 1953. He received the B.Sc., M.Sc., and Ph.D. degrees from National Chiao Tung University, Republic of China, in 1976, 1980, and 1990, respectively, all in computer science and information engineering.

He joined Honeywell Company in Taiwan in 1979. He became a Technical Manager there in 1981, involved in the design and implementation of the advanced industrial control systems (1981-1986) and the intelligent enterprise management systems (1986-1990). From 1982 to 1984 he was also a part-time lecturer in the Department of Electronics Engineering and Technology at National Taiwan Institute of Technology. In 1991, he joined, as the Vice President, K&C Technologies, Inc., Taiwan. His research interests include parallel algorithm and architecture design, computer vision, pattern recognition, and artificial intelligence.

Dr. Lin received in 1976 the Phi-Tau-Phi Award for best student in his class. He also received the Outstanding Manager Award from Honeywell in 1986. He is a member of ASHARE, the Chinese Image Processing and Pattern Recognition Society, and the Chinese Engineering Society.

Zen Chen received the B.Sc. degree from National Taiwan University, Taiwan, Republic of China, in 1967, the M.Sc. degree from Duke University, Durham, NC, in 1970, and the Ph.D. degree from Purdue University, West Lafayette, IN, in 1973, all in electrical engineering.

From 1973 to 1974 he worked for Burroughs Corporation, Detroit, MI, where he was engaged in the development of document recognition systems. Since 1974 he has taught at the Institute of Computer Science and Information Engineering, National Chiao Tung University, Taiwan. He was the Director of the Institute from 1975 to 1981, and from 1989 to 1991. He spent the academic year 1981 to 1982 at Lawrence Berkeley Laboratory, University of California, Berkeley, as a Visiting Scientist and also spent six months on sabbatical (starting from August 1989) at the Center for Automation Research, University of Maryland at College Park, as a Visiting Professor. His areas of interest include computer vision, pattern recognition, expert systems, parallel algorithm and architecture, computer graphics, and CAD/CAM systems.

He is a member of Sigma Xi, Phi Kappa Phi, and Phi Tau Phi.