MATRIX FORMULATION AND PARALLEL ALGORITHMS FOR RELAXATION OBJECT LABELING

Eva Leung and Xiaobo Li

Department of Computing Science
University of Alberta
Edmonton, Alberta, T6G 2H1

ABSTRACT

Object labeling by relaxation is a very powerful tool in iteratively reducing local ambiguities by employing contextual information. Applications of relaxation labeling cover a diversity of different disciplines ranging from scene analysis to graph isomorphism. This paper presents a generalized matrix formulation of relaxation object labeling including the discrete, the fuzzy and the linear stochastic model, upon which two parallel algorithms on SIMD machines with different interconnection networks and different number of PEs (processing elements) are presented. The generalized model transforms the labeling problem into a matrix/vector "multiplication" problem. The "multiplication" is defined depending on specific model used. The labeling problem involves the matching of M labels to N objects. Time complexity of the algorithm is $O(N^2M^2)$ per iteration on a sequential machine. The first parallel algorithm is written on a mesh connected machine with boundary end-around using NM PEs. Time complexity of the algorithm is $O(NM)$. The second algorithm is developed on hypercube connected machine with N^2M^2 PEs. The complexity of the second algorithm is $O(\log M + \log N)$.

KEYWORDS : Vision architecture, Relaxation object labeling, SIMD computers, Parallel algorithm, Interconnection network.

I. INTRODUCTION

Relaxation labeling processes are a class of iterative algorithms which reduce local ambiguities by using contextual information. Relaxation techniques have been found to be very useful and attractive for reducing labeling errors in various types of image data and their performance is remarkably well in many domains. For instance, the relaxation labeling of simple tasks, such as labeling the sides of a triangle [7, 13], has demonstrated that perfect labeling is possible. Not only has the labeling problem been shown to play an important role in both scene analysis and computer vision [3, 7, 11], it is also a generalization of several specific problems belonging to different areas: the subgraph isomorphism problem, the graph homomorphism problem, the automata homomorphism problem, the packing problem, the shape matching problem [3].

Several algorithms [7, 8, 12] have been proposed for modifying an initial estimate of the labeling of a scene element by reference to spatial context. Many of them are actually variations of the relaxation techniques devised by Rosenfeld *et al* [8, 12]. A total of four different models are discussed in [7], namely, the discrete, the fuzzy, the linear probabilistic and the nonlinear probabilistic models. In this paper, a generalized matrix representation for various models is introduced which can readily be implemented on parallel processing machines. With sequential machine, the time complexity per iteration is $O(N^2M^2)$ which can be reduced to $O(NM)$ if parallelism is exploited.

In the next section, we use a generalized matrix form for the first three relaxation models defined in [7]. A SIMD machine model is described in section III. Section IV gives an algorithm for mesh network with boundary end-around connection, i.e., the torus network, using NM PEs. An alternative to mesh is the hypercube network which is described together with an algorithm working on N^2M^2 PEs in section V. Lastly, a conclusion is given in section VI.

II. MATRIX FORMULATION

In this section, we use a generalized matrix form for several different relaxation labeling models, mainly Rosenfeld's discrete model, fuzzy model and linear stochastic model [7]. Let $A=\{a_1, \ldots, a_N\}$ be a set of N objects and $\Lambda=\{\lambda_1, \lambda_2, \ldots, \lambda_M\}$ be a set of M labels where M and N are integers. The objective of labeling is to assign a unit label, say λ_α, α in $\{1, 2, \ldots, M\}$, to each object a_i, i in $\{1, 2, \ldots, N\}$, in an input image. We use $a_i=\lambda_\alpha$ to represent the assignment of label λ_α to object a_i. A mapping, $l_i:\Lambda \rightarrow Range$, defines a labeling on object a_i where the range of l_i, *Range*, depends on the particular model employed. In the discrete model, *Range* is the set $\{0,1\}$, and $l_i(\lambda_\alpha)=1$ indicates that λ_α is a legal (allowed) label for object a_i, while $l_i(\lambda_\alpha)=0$ means

The authors are supported in part by the Canadian National Science and Engineering Research Council under Grant A9198 and the Central Research Fund of University of Alberta.

that it is impossible to label a_i with λ_α. In the fuzzy model and the linear probabilistic model, *Range* is the closed interval [0,1]; and $l_i(\lambda_\alpha)$ represents the degree of confidence or the probability that a_i can be labeled with λ_α. We use superscripts to denote iterations. The notation $l_i^{(k)}(\lambda_\alpha)$ represents our belief about labeling a_i with λ_α at the k-th iteration, while $k=0$ is the initial labeling restriction obtained from a priori knowledge.

The labeling vector $L_i^{(k)}$ for object a_i is defined below:

$$L_i^{(k)} = \begin{bmatrix} l_i^{(k)}(\lambda_1) \\ l_i^{(k)}(\lambda_2) \\ \vdots \\ l_i^{(k)}(\lambda_M) \end{bmatrix}_{M\times 1}$$

Based on the $L_i^{(k)}$'s, a long column vector $L^{(k)}$ is defined below to represent the labeling of all N objects at iteration k:

$$L^{(k)} = \begin{bmatrix} L_1^{(k)} \\ L_2^{(k)} \\ \vdots \\ L_N^{(k)} \end{bmatrix}_{NM\times 1}$$

Another mapping, $c_{ij}: \Lambda\times\Lambda\rightarrow Range$, defines the effect of object a_j on object a_i. The range of c_{ij}, *Range*, is again dependent on the particular model employed. The meaning of $c_{ij}(\lambda_\alpha, \lambda_\beta)$ in each model is listed below:

In the discrete model,

$c_{ij}(\lambda_\alpha, \lambda_\beta)=1$: $a_i=\lambda_\alpha$ is compatible with $a_j=\lambda_\beta$;

$c_{ij}(\lambda_\alpha, \lambda_\beta)=0$: $a_i=\lambda_\alpha$ is not compatible with $a_j=\lambda_\beta$.

In the fuzzy model,

$c_{ij}(\lambda_\alpha, \lambda_\beta)$ is the confidence of assigning λ_α to a_i given the assignment of λ_β to a_j.

In the linear stochastic model,

$c_{ij}(\lambda_\alpha, \lambda_\beta)$ is the conditional probability of assigning λ_α to a_i given the assignment of λ_β to a_j.

We now define the compatibility matrix C_{ij} (the effect of the labeling of a_j to the labeling of a_i) as below:

$$w_{ij}\begin{bmatrix} c_{ij}(\lambda_1,\lambda_1) & c_{ij}(\lambda_1,\lambda_2) & \cdots & c_{ij}(\lambda_1,\lambda_M) \\ c_{ij}(\lambda_2,\lambda_1) & c_{ij}(\lambda_2,\lambda_2) & \cdots & c_{ij}(\lambda_2,\lambda_M) \\ \vdots & \vdots & \vdots & \vdots \\ c_{ij}(\lambda_M,\lambda_1) & c_{ij}(\lambda_M,\lambda_2) & \cdots & c_{ij}(\lambda_M,\lambda_M) \end{bmatrix}_{M\times M}$$

where w_{ij} is a weighting factor specifying the importance of a_j to a_i regardless the particular labels involved. Different definitions of w_{ij}'s are given in Table 1.

Table 1. Definition of the functions, ranges and weighting factors in the generalized form for different labeling models.

model	h	g	f	*Range*	w_{ij}
discrete	$\min_j$	$\max_\beta$	$\times$	{0, 1}	$w_{ij}=1$
fuzzy	$\min_j$	$\max_\beta$	min	[0, 1]	$w_{ij}=1$
linear stochastic	weighted sum over j	$\sum_\beta$	$\times$	[0, 1]	$\sum_j w_{ij}=1$

Now, a large compatibility matrix for the entire object set is defined as:

$$C = \begin{bmatrix} C_{11} & C_{12} & \cdots & C_{1N} \\ C_{21} & C_{22} & \cdots & C_{2N} \\ \vdots & \vdots & & \vdots \\ C_{N1} & C_{N2} & \cdots & C_{NN} \end{bmatrix}$$

$$= \begin{bmatrix} c_{11}(\lambda_1,\lambda_1) & \cdots & c_{1N}(\lambda_1,\lambda_M) \\ \vdots & \cdots & \vdots \\ \vdots & \cdots & \vdots \\ c_{N1}(\lambda_M,\lambda_1) & \cdots & c_{NN}(\lambda_M,\lambda_M) \end{bmatrix}_{NM\times NM}$$

For convenience, we use $C[x,y]$ to denote the (x,y)-th entry of the large matrix C. This entry is related to $c_{ij}(\lambda_\alpha, \lambda_\beta)$ with the following equations:

$$x = (i-1)M+\alpha$$

and

$$y = (j-1)M+\beta$$

We now proceed to define the relaxation operation, i.e., the iterative modification of a labeling of object a_i, as the following computation of vector L_i:

$$l_i^{(k+1)}(\lambda_\alpha) = \underset{j}{h}\left[\underset{\beta}{g}\left[f\left[c_{ij}(\lambda_\alpha,\lambda_\beta), l_j^{(k)}(\lambda_\beta)\right]\right]\right] \qquad (1)$$

where functions f, g and h are defined in Table 1. The labeling of the entire object set, defined as a function of previous iteration, is given in matrix form:

$$L^{(k+1)}[x] = h\left[g\left[f\left[C[x,y], L^{(k)}[y]\right]\right]\right]$$

The relaxation labeling problem is thus reduced to a matrix/vector multiplication problem where the multiplication is defined as a combination of the three functions f, g and h,

$$L^{(k+1)} = CL^{(k)}$$

Notice that the f takes precedence over both g and h; and with the exception to the probabilistic model (in which g and h are the same), g takes precedence over h. Because the combination of the three functions is in general non-commutative, hence, special attention is taken in all the parallel algorithms to enforce precedence in the operations of the functions.

III. MODEL OF AN ARRAY PROCESSOR

An array processor is composed of Q PEs, indexed 1 through Q, for some integer Q. The PEs form a conceptual 2-D array of size $Q_1 \times Q_2$, for some integers Q_1, Q_2. PE(p) with $1 \le p \le Q$ is also referred to as PE(x,y) where $p = (x-1)Q_2 + y$, with $1 \le \rho \le Q_1$ and $1 \le \tau \le Q_2$. Each PE has a local memory of size O(MN) and several registers.

The array processors discussed in this paper are synchronized SIMD computers. The PEs are synchronized and execute instructions issued from a control unit. The control unit broadcasts an instruction to all PEs, and all enabled PEs simultaneously execute the instruction. The enable/disable mask can be used to select a subset of the PEs that are to perform an instruction. The set of enabled PEs can be changed from instruction to instruction. No synchronization primitives are needed during computation. Parallel algorithms on SIMD machines pre-schedule all PEs on the systems. A communication network, mesh with boundary end-around connections or hypercube, is used to provide inter-PE communications.

In describing our algorithms, we follow the notation common in the literatures [1, 2]. The symbol "$\leftarrow$" denotes an assignment involving data routing between directly connected PEs; ":=" denotes local assignment in one PE; while "$\Leftarrow$" denotes an assignment requiring common data or constant movement from the control unit memory to PE registers. For example, with mesh network, if we want to transfer information from registers $R1$ in PEs with odd row numbers up to those with even row numbers, the statement will be

$$R1(x+1,y) \leftarrow R0(x,y) \qquad (x_1 = 1)$$

The condition in parentheses after the assignment is the enable mask. Consider $x = x_n x_{n-1} \ldots x_1$ to be the n-bit binary representation of the row index with x_n to be the most significant bit (MSB) and x_1 the least significant bit (LSB). If the LSB of the row index of the PE is 1, it is enabled; otherwise, disabled. If no mask function is provided, all PEs are enabled.

IV. PARALLEL ALGORITHM ON SIMD MACHINE WITH TORUS NETWORK

Mesh interconnection network is the most popular network for SIMD machines. In a mesh network, each PE is connected to its four immediate neighbors (above, below, left and right). Different versions of mesh network have different setup for the connection of the boundary PEs. This section proposes a parallel procedure for one iteration of relaxation labeling on array processor with end-around mesh, which is also know as a torus. NASA's MPP is capable of being connected as a torus for it can be programmed to have the left ens of each row and column connected to the right end of the same row and column respectively [5].

A mesh network could be in multi-dimensions. If PEs are arranged as in a k-D array, $Q = Q_k \times Q_{k-1} \times \cdots \times Q_1$. PE($p_k, \cdots, p_1$) is connected to PE($p_k, \cdots, p_x \pm 1, \cdots, p_1$) for $1 \le x \le k$. Each PE has at most $2k$ links. In this section, we consider 2-D mesh only and the routing strategy is

$R1(x+1,y) \leftarrow R1(x,y)$ (* up *)
$R1(x-1,y) \leftarrow R1(x,y)$ (* down *)
$R1(x,y-1) \leftarrow R1(x,y)$ (* left *)
$R1(x,y+1) \leftarrow R1(x,y)$ (* right *)

Section II presented a formal matrix formulation of the generalized model. However, from an operational point of view, some elements of C are redundant and the C matrix can be compressed horizontally to form a matrix of order $NM \times M(N-1)$. Notice that all C_{ii}'s are diagonal matrix of the form

$$C_{ii} = w_{ii} \begin{bmatrix} 1 & 0 & \ldots & 0 \\ 0 & 1 & \ldots & 0 \\ \vdots & \vdots & & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ & & \vdots & \\ 0 & 0 & \ldots & 1 \end{bmatrix}_{M \times M}$$

and $g_{\beta}\left[f\left[c_{ii}(\lambda_\alpha, \lambda_\beta), l_i^{(k)}(\lambda_\beta)\right]\right]$ yields $w_{ii} l_i$ for all the three models. So all submatrices C_{ii} can be deleted from C to give a horizontally compressed C matrix,

$$CC = \begin{bmatrix} C_{12} & C_{13} & \ldots & C_{1N} \\ C_{21} & C_{23} & \ldots & C_{2N} \\ \vdots & \vdots & & \vdots \\ C_{N1} & C_{N2} & \ldots & C_{N(N-1)} \end{bmatrix}$$

Equation (1) can be modified and $l_i^{(k+1)}(\lambda_\alpha)$ becomes

$$h\left[w_{ii} \times l_i^{(k)}(\lambda_\alpha),\ \underset{j \ne i}{h}\left[g_{\beta}\left[f\left[c_{ij}(\lambda_\alpha, \lambda_\beta),\ l_j^{(k)}(\lambda_\beta)\right]\right]\right]\right]$$

Originally, register $R1$ of PE(x,y) stores $[(x-1)M+y]$-th element of the long vector L, and $R4$ the w_{ii} where $i = \left\lfloor \frac{x-1}{M} \right\rfloor + 1$. Row $(x-1)M+y$ of the CC matrix is stored in the local memory of PE(x,y) in a skewed fashion. An example for $N=3$ and $M=5$ is given in Fig.1 with the contents of the local memory listed for each PE.

Note that each PE is listed according to its position in the 2D array of $N \times M$, and the locations are counted from 1 instead of 0. The placement of the C matrix in the local memory of each PE is very much dependent on the how data are being transferred, hence, we will delay the discussion and move on to describe the algorithm first. Now, we will just assume that the skewed memory always enable f to be applied to correct pairs of arguments.

The first step of computation is the initialization of $R3$ by multiplying $R1$ with $R4$. The contents in $R1$ are then shifted up to the PEs immediately above. Function f is applied to register $R1$ and a word in the local memory. The contents in registers $R1$ are then shifted to the right $M-1$ times. After each shift, the computation of $g(f)$ is performed between $R1$ and the corresponding word in local memory. Hence, at the end of the horizontal shifts, we obtain the result in applying g over M $f[C,L]$ in $R2$. Again, the contents in $R1$ are shifted one step up and function h is performed between $R2$ and $R3$. Once again, we perform the $M-1$ shift and obtain a new $g(f)$ result in $R3$. The above shift up/right process is repeated until after $N-1$ vertical shifts, each new $L[p]$ is computed in PE(p). The skewed storage scheme makes the updating of MAR very simple, just increment. Each L element appears at the right PEs (PEs which need that particular element for computa-

tion) exactly once, and always meet with the correct C element to perform f operation. The procedure is shown below:

```
Procedure Torus
begin
    MAR := 0
    R3 := R1*R4
    for w=1 to N-1 do
        R1(x-1, y)<-R1(x, y)   (* shift up *)
        MAR := MAR+1
        R2 := f(R1, mem)
        for z=1 to M-1 do
            R1(x, y+1)<-R1(x, y)   (* shift right *)
            MAR := MAR+1
            R2 := g(R2, f(R1, mem))
        endfor
        R3:=h(R3, R2)
    endfor
    R1:=R3
end (* torus *)
```

This procedure requires $(N-1)M$ communication steps. It performs the function f $(N-1)M$ times, g function $(N-1)(M-1)$ times, and h function $N-1$ times. The total time complexity is $O(NM)$.

We will now consider ordering of the elements in each row of C within the local memory of each PE. The labeling formula for one iteration is basically the application of combined function hgf on each pair of $C[x,y]$ and $L[y]$ where x,y are indices. Initially, we store $L[y]$ in PE(y) and the x-th row of C in PE(x). The main idea of the procedure is to ship $L[y]$ to wherever the $C[x,y]$'s reside and perform whatever operation required. We can define the distance between $C[x,y]$ and $L[y]$ in terms of the number of right shifts, *rt*, and the number of upward shifts, *up*, required. Mathematically, we have

$$up = \left(\left\lfloor \frac{x}{M} \right\rfloor - \left\lfloor \frac{y}{M} \right\rfloor \right) \bmod N$$

and

$$rt = (x-y) \bmod M$$

Hence the location, *loc*, of $C[x,y]$ in PE(x) is equal to the total number of steps required to bring $L[y]$ to PE(x). Therefore, *loc* is equal to the vertical/horizontal steps needed in order to shift $L[y]$ to row (x div M) plus horizontal steps needed to shift to PE(x), which is

$$(up-1)(M-1)+up+\left[rt - \left[((up-1)(M-1)) \bmod M\right] \bmod M\right]$$

which when simplified, *loc* becomes

$$1 + (up-1)(M) + (rt+up-1) \bmod M$$

If $CC[x,y']$ is used then we have

$$up = \left(\left\lfloor \frac{x}{M} \right\rfloor - \left\lfloor \frac{y'}{M} \right\rfloor \right) \bmod N + 1$$

and

$$rt = (x-y') \bmod M$$

The corresponding *loc* can be calculated with the same formula.

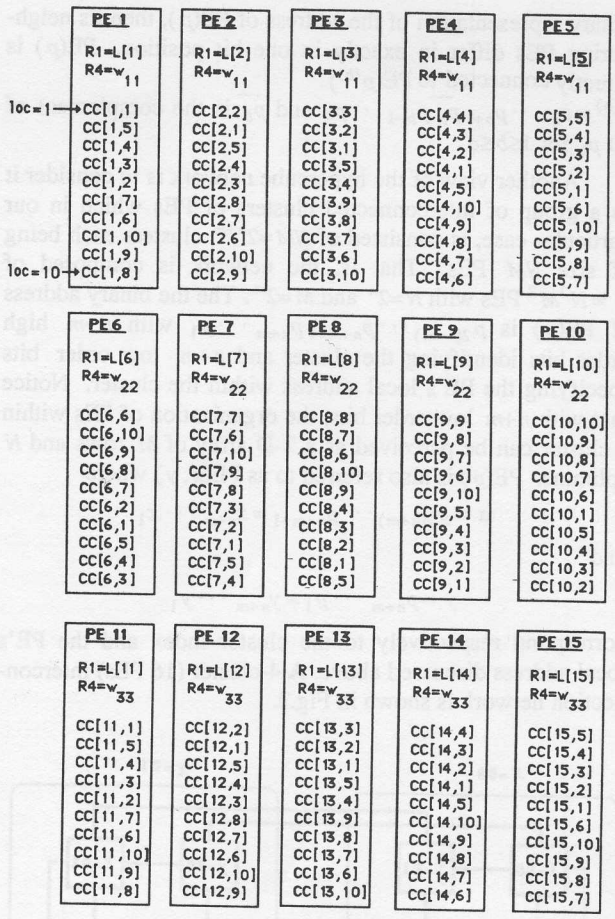

Fig. 1. Initial contents of registers and local memory of each PE with N=3 and M=4.

A further attempt in exploiting additional parallelism can be made by considering an array processor with N^2M^2 PEs. However, an inherent property of the g and h functions requires the accumulation/comparison of all the intermediate results on each row of C so that one final $l_i(\lambda_\alpha)$ can be obtained. The acquisition and the re-distribution of $l_i(\lambda_\alpha)$'s among the PEs increase the communication cost if more PEs are used. Using N^2M^2 PEs can reduce the computation of f by a factor of NM while at the same time increase the communication cost also by a factor of NM. Hence, the tradeoff is apparently between the computation cost of f and the communication cost of the PEs.

V. HYPERCUBE INTERCONNECTION NETWORK WITH N^2M^2 PEs

This section provides a procedure of one iteration of relaxation on a hypercube machine. Hypercube network has been proposed and used in both SIMD and MIMD computer system [6, 9]. Many parallel processors based on hypercube topology are available in the market, such as the Connection Machine [4], Intel's iPSC and Ncube [10].

The hypercube network is also known as a q cube network, if $Q=2^q$ is the number of PEs available, for it is an extension to the unit-cube concept. With hypercube, the addresses of PEs start from 0 to $Q-1$. In the q cube, each PE is directly connected to q neighbors. If $p = p_q \cdots p_1$ is the

binary representation of the address of PE(p), then its neighboring PEs differ in exactly in one bit position. PE(p) is directly connected to PE($p^{(b)}$)
$p^{(b)}=p_q \cdots p_{b+1}\,\overline{p_b}\,p_{b-1}\cdots p_1$ and $\overline{p_b}$ is the complement of bit p_b for $1\le b\le q$.

Another view of the hypercube network is to consider it as a group of interconnected clusters of PEs which in our particular case, is consisted of $NM=2^{n+m}$ clusters each being of size NM PEs. That is, the network is composed of $Q=N^2M^2$ PEs with $N=2^n$ and $M=2^m$. The the binary address of PE(p) is $p_{2(n+m)}\cdots p_{n+m+1}p_{n+m}\cdots p_1$ with $n+m$ high order bits identifying the cluster and $n+m$ low order bits specifying the PE's local address within the cluster. Notice that with $n+m$ low order bits, the organization of PEs within a cluster can be perceived as a 2-D array of M rows and N columns. PE(p) is also referred to as PE(x,y) where

$$x=p_{2(n+m)}\cdots p_{n+m+1}=x_{n+m}\cdots x_1$$

and

$$y=p_{n+m}\cdots p_1=y_{n+m}\cdots y_1$$

correspond respectively to the cluster index and the PE's local address discussed above. A 4-cluster (16 PEs) interconnection network is shown in Fig.2.

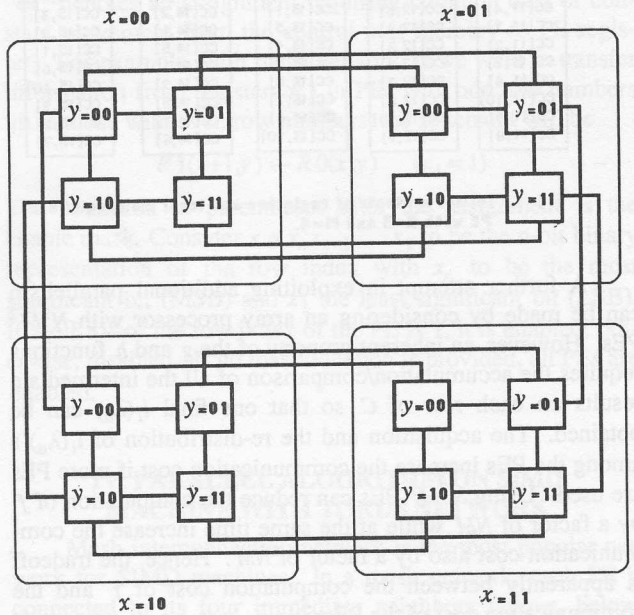

Fig. 2. A 4-cluster hypercube network. x is the cluster number and y is the local address of PE(x,y) within the cluster.

The central idea of the implementation is to have each cluster responsible for the updating of one possible labeling, a single element of L. Therefore, it is necessary to store replicas of L into each cluster together with the corresponding row vector of the C matrix. Initially, each cluster contains an element of L in the PE which has its local address being the same as its cluster index, i.e. $L[x-1]$ in PE(x,x). The $l_i(\lambda_\alpha)$'s are then distributed over to all clusters simultaneously so that with PE(x,y), the contents of registers $R1$ and $R2$ are $L(y+1)$ and $C[x+1,y+1]$ respectively. Then, the N^2M^2 PEs compute all the $f(C,L)$ in one step. Functions g and h are performed on individual clusters with all clusters operating in parallel. Function g is being applied to rows followed by h to columns. The cluster index, x, is employed as the destination index of the resulting $l_i(\lambda_\alpha)$. In other words, at the end of all computations, each updated $l_i(\lambda_\alpha)$ is again stored in PE with local address being equal to its own cluster index. This is easily achieved by having the cluster index served as a mask for routing during computation. Below is procedure *q−cluster* for the updating of L in one iteration.

```
Procedure q-cluster
begin
    for s=1 to n+m do
        R1(p^(n+m+s))←R1(p)  (x_{n+m}..x_s=y_{n+m}..y_s)
    endfor
    R1:=f(R1,R2)
    for s=1 to m do
        R3(p^(s))←R1(p)   (x_s=y_s)
        R1:=g(R1,R3)
    endfor
    for s=m+1 to m+n do
        R3(p^(s))←R1(p)   (x_s=y_s)
        R1:=h(R1,R3)
    endfor
end; (* q-cluster *)
```

This procedure requires one step for computing f, m steps for g while n steps for h. A total of $2(n+m)$ steps are used for communications between PEs. Hence, the procedure requires a total of $3(n+m)+1$ steps. Time complexity is $O(\log N+\log M)$.

VI. CONCLUSION

A matrix formalization of three different relaxation labeling model is presented in this paper. In this matrix format, a single algorithm can be used for all three models. This is useful and attractive in the practice of image understanding. In the parallel processing environment, this formalization simplifies the data preparation and manipulation into a single parallel algorithm. This matrix representation may take advantages of other standard matrix multiplication algorithms developed on different machines. The inclusion of other relaxation labeling models into this matrix formalization is current under investigation.

This paper proposed two parallel labeling algorithms on SIMD machines with Torus mesh network and hypercube network. The complexity of the algorithm is $O(NM)$ for MN PEs with mesh, and $O(\log N+\log M)$ with hypercube, while the complexity of a sequential algorithm is $O(N^2M^2)$.

References

1. E. Dekel, D. Nassimi and S. Sahni, "Parallel matrix and graph algorithms", *SIAM J. Comp.*, Vol. 10, No. 4, 1981, pp. P 657-675.
2. Z. Fang, X. Li and L. Ni, Parallel Algorithms for image template matching on hypercube SIMD computers, *IEEE CAPIDM Workshop*, 1985, pp. 33-40.
3. R. M. Haralik and L. G. Shapiro, "The consistent labeling problem : Part I", *IEEE Transactions on Pattern Analysis and Machine Intelligence,* Vol. PAMI-1, No. 2, April 1979, pp. 173-184.
4. W. D. Hillis, in *The Connection Machine*, MIT Press, Cambridge, MA, 1985.
5. K. Hwang and F. A. Briggs, in *Computer Architecture and Parallel Processing*, McGraw-Hill, 1984, pp. 422-426.
6. F. Preparata and J. Vuillemin, The cube-connected cycles: A versatile network for parallel computation, *Proc. IEEE Symp. on Foundations of Computer Science*, 1979, pp. 140-144.
7. A. Rosenfeld, R. A. Hummel and S. W. Zucker, "Scene labeling by relaxation operations", *IEEE Transactions on Systems, Man, Cybernetics,* Vol. SMC-6, No. 6, June 1976, pp. 420-433.
8. W. R. Rutkowski, S. Peleg and A. Rosenfeld, "Shape segmentation using relaxation", *IEEE Transactions on Pattern Analysis and Machine Intelligence,* Vol. . PAMI-3, No. 4, July 1981, pp. 368-375.
9. C. Seitz, "The cosmic cube", *Comm. of the ACM,* Vol. 28(1), January 1985, .
10. P. Wallich and G. Zorpette, Minis and mainframes, *IEEE Spectrum*, January 1986, pp. 36-39.
11. D. L. Waltz, Understanding line drawings of scenes with shadows, in *The Psychology of Computer Vision*, P. H. Winston (ed.), McGraw-Hill, New York, 1975, pp. 19-91.
12. H. Yamamoto, "A method of deriving compatibility coefficients for relaxation operations", *Comp. Graph. Image Processing,* Vol. 10, 1979, pp. 256-271.
13. S. Zucker, E. Krishnamurthy and R. Haar, "Relaxation processes for scene labeling: Convergence, speed and stabilility", *IEEE Transactions on Systems, Man, Cybernetics,* Vol. SMC-8, Jan 1978, pp. 41-48.