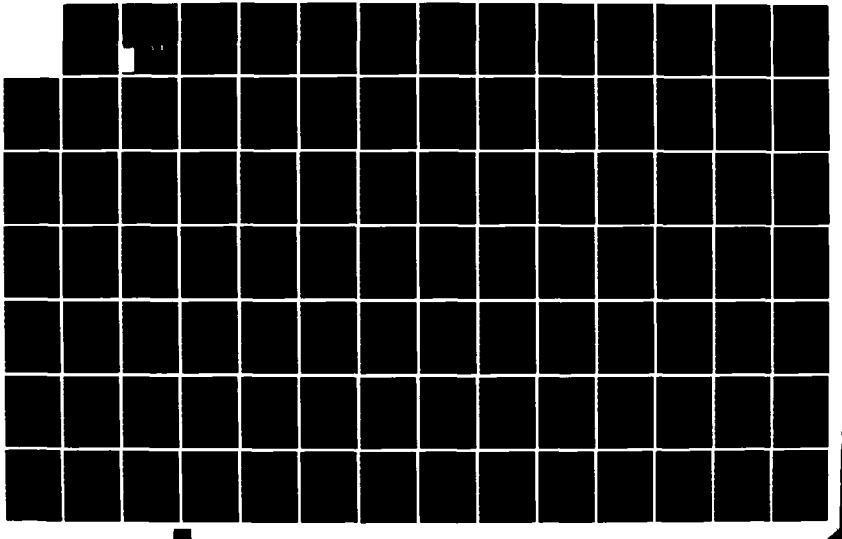
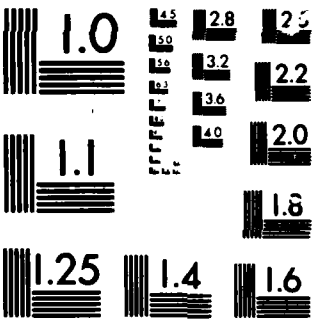
AD-A167 317 DISTRIBUTED COMPUTING FOR SIGNAL PROCESSING: MODELING OF ASYNCHRONOUS PAR..(U) PURDUE UNIV LAFAYETTE IN SCHOOL OF ELECTRICAL ENGINEERING G LIN ET AL. AUG 84 1/2

UNCLASSIFIED TR-EE-84-29 ARO-18790.17-EL-APP-F F/G 9/2 NL





MICROCOPY CHART

Ph.D. Thesis by: Gie-Ming Lin*
Faculty Advisor: Philip H. Swain
APPENDIX F for
Distributed Computing for Signal
Processing: Modeling of Asynchronous
Parallel Computation; Final Report
for U.S. Army Research Office
Contract No. DAAG29-82-K-0101
*Chapter 3 supported by this contract

(2)

Studies in Parallel Image Processing

Gie-Ming Lin
Philip H. Swain

TR-EE 84-29
August 1984

School of Electrical Engineering
Purdue University
West Lafayette, Indiana 47907

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Studies in Parallel Image Processing		5. TYPE OF REPORT & PERIOD COVERED Technical Report
		6. PERFORMING ORG. REPORT NUMBER TR-EE 84-29
7. AUTHOR(s) Gie-Ming Lin, Philip H. Swain		8. CONTRACT OR GRANT NUMBER(s) Contract nos. F30602-83-K-0119, DAAG29-82-K-0101
9. PERFORMING ORGANIZATION NAME AND ADDRESS School of Electrical Engineering Purdue University West Lafayette, IN 47907		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Rome Air Development Center, USAF; Army Research Office.		12. REPORT DATE August 1984
		13. NUMBER OF PAGES 163
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

N/A

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

image processing, parallel processing, pattern classification, remote sensing.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The supervised relaxation operator combines the information from multiple ancillary data sources with the information from multispectral remote sensing image data and spatial context. Iterative calculations integrate information from the various sources, reaching a balance in consistency between these sources of information. The supervised relaxation operator is shown to produce substantial improvements in classification accuracy compared to the accuracy produced by the conventional

maximum likelihood classifier using spectral data only. The convergence property of the supervised relaxation algorithm is also described.

Improvement in classification accuracy by means of supervised relaxation comes at a high price in terms of computation. In order to overcome the computation-intensive problem, a distributed/parallel implementation is adopted to take advantage of a high degree of inherent parallelism in the algorithm. To accomplish this, first, a graphical modeling and analysis method is decribed for algorithms implemented using the SIMD mode of parallelism. Second, the comparison of execution times between SIMD and MIMD modes is discussed, based on the analysis of implicit and explicit operations embedded in an algorithm. From the comparison it is shown that some algorithms are suitable for MIMD mode of parallelism; some are more suited to SIMD. Third, several performance measures for SIMD, functions of problem size and system size, are discussed. Finally, two multistage interconnection networks to support the distributed/parallel system are overviewed. Based on these considerations, an optimal system configuration in terms of execution time and system cost is proposed.

STUDIES IN PARALLEL IMAGE PROCESSING

Gie-Ming Lin

Philip H. Swain

Purdue University

School of Electrical Engineering

West Lafayette, IN 47907

USA

TR-EE 84-29

August 1984

This research was supported by the United States Air Force Command, Rome Air Development Center, under contract no. F30602-83-K-0119, and by the U.S. Army Research Office, Department of the Army, under contract no. DAAG29-82-K-0101. Approved for public release, distribution unlimited.

TABLE OF CONTENTS

CHAPTER 1 - INTRODUCTION

The conventional classification of multispectral image data collected by remote sensing devices such as the multispectral scanners aboard the Landsat or Skylab sateilites has usually been performed such that each pixel is classified by using spectral information independent of the neighboring pixels. There is no provision for using the spatial information inherent in the data. In many cases there are available other sources of data which an analyst can use as well as spatial information to establish a context for deciding what a particular pixel in the imagery might be. By utilizing this contextual information, it may be possible to achieve an improvement in classification accuracy. For example, in the case of forestry, various tree species are known to exhibit growth patterns dependent upon topographic position. When this fact is used along with spectral and spatial information, a classification with enhanced accuracy can be obtained [1,2].

The supervised relaxation operator which combines information from spectral, spatial, and ancillary data to classify multispectral image data is part of the subject of this report. Relaxation operations are a class of iterative, parallel techniques for using contextual information to reduce local ambiguities [3]. Such techniques have been found to be useful in many applications such as edge reinforcement [4], image noise reduction [5], histogram modification, thinning, angle detection, template matching, and region labelling [6]. Its

convergence, the derivation of compatibility coefficients, and the theoretical foundations of relaxation labeling processes have been described in [7,8,9,10,11]. A modification to existing probabilistic relaxation processes to allow the information contained in the initial labels to exert an influence on the direction of relaxation throughout the process has been described in [12]. This modified relaxation method, called supervised relaxation labeling, has been extended to incorporate one ancillary information source into the results of an existing classification [2]. In this thesis, a method for integrating image data from multiple ancillary data sources is described in Chapter 2. Based on the approach in [13], the convergence property of the supervised relaxation operator is presented. The supervised relaxation operator is generalized and demonstrated experimentally to combine information from spatial and multiple ancillary data sources with the spectral information for the classification of multispectral imagery with multiple classes.

Due to the high computational complexity of such operations and the availability of low cost microprocessors, architectures involving multiple processors are very attractive. Several parallel organizations have been proposed, principally SIMD and MIMD architectures. In Chapter 3 of this report we focus our attention on performance measures for parallel processors and interconnection networks for multiprocessor systems. Section 3.1 will present the application of S-Nets [14,15] to modeling the maximum likelihood algorithm in SIMD and pipeline implementations. Several alternative performance measures for the parallelism inherent in the algorithm are also described. The algorithm execution times derived based on the analysis of implicit and explicit operations in the algorithm for SIMD and MIMD modes of parallelism are compared and discussed in Section 3.2. In Section 3.3, the

determination of the optimal number of processing elements in terms of execution time and system cost for a given image size based on the measures of evaluating the performance of an algorithm is discussed. Finally, Section 3.4 describes how two multistage networks [16,17,18,19], the cube network and ADM network, can be configured to support an implementation which utilizes both SIMD and MIMD processing components.

CHAPTER 2 - SUPERVISED RELAXATION OPERATOR

In this chapter, Section 2.1 contains a step-by-step description of the supervised relaxation operator. In Section 2.2, a method for the integration of image data from multiple auxiliary data sources will be given. Section 2.3 discusses the convergence property of the supervised relaxation algorithm. Finally, Section 2.4 presents experimental results on a 182-by-182 pixel Landsat image.

2.1 Derivation of the Supervised Relaxation Algorithm

In this section, the derivation of the supervised relaxation operator from the original relaxation operator [3] is described step by step, and an heuristic interpretation of the supervised relaxation operator is given.

2.1.1 Probabilistic Relaxation Operator

Most classifiers used with remote sensing data are pixel-specific. Each pixel is classified by using spectral information independent of the classification of the neighboring pixels. No spatial or contextual information is used.

Relaxation labeling processes are a class of iterative, parallel techniques for using contextual information to disambiguate probabilistic labelings of objects [3]. They make use of two different sources of information, a priori neighborhood models and initial observations, which interact to produce the

final labelings. The relaxation processes involve a set of objects, $A = \{a_1, a_2, ..., a_n\}$, and the relationship of each object to its neighbors. Attached to each of the objects is a set of labels, $\Lambda = \{\lambda_1, \lambda_2, ..., \lambda_m\}$, where each label indicates a possible interpretative assertion about that object; for example, the objects might be image pixels and the labels might be spectral class names. The relaxation algorithm attempts to use constraint or compatibility relationships defined over pairs of labels, possible interpretations, attached to current and neighboring objects in order to eliminate inconsistent combinations of labels. ("Current" refers to an object which, at the moment, is the focus of attention.)

A measure of likelihood or confidence is associated with each of the possible labels attached to objects. This measure is denoted by $P_i(\lambda)$. These likelihoods satisfy the condition

$$\sum_{\lambda \epsilon \Lambda} P_i(\lambda) = 1, \text{ for all } a_i \epsilon A, 0 \leq P_i(\lambda) \leq 1.$$

In this probabilistic model, the likelihoods estimated for an object's labels are updated on the basis of the likelihoods distributed among the labels of the neighboring objects. These likelihoods interact through a set of compatibility coefficients that are defined for each pair of labels on current and neighboring objects. The compatibility coefficients are determined a priori based on some foreknowledge of or on a "typical" model for the image to be classified. More specifically, for a given object a_i, the likelihood $P_i(\lambda)$ of a given label λ should increase if the neighboring objects' labels with high likelihoods are highly compatible with λ at a_i. Conversely, $P_i(\lambda)$ should decrease if neighboring high-likelihood labels are incompatible with λ at a_i. On the other hand, neighboring labels with low likelihoods should have little influence on $P_i(\lambda)$ regardless of

their compatibility with it. Let λ' be a label for a neighboring object and λ for the current object. These characteristics can be summarized in tabular form as follows:

		Compatibility of λ' with λ	
		High	Low
Likelihood of λ'	High	+	−
	Low	0	0

where + means that $P_i(\lambda)$ should increase, − means that it should decrease, and 0 means that it should remain relatively unchanged. The coefficient $\gamma_{ij}(\lambda,\lambda')$ is a measure of the compatibility between label λ on current object a_i and label λ' on neighboring object a_j. These compatibility coefficients are defined over the range $[-1, 1]$. The aim is that the γ's should behave as follows:

$\gamma_{ij}(\lambda,\lambda') > 0$, if λ on a_i frequently co-occurs with λ' on a_j.

$= 1$, if λ on a_i always co-occurs with λ' on a_j.

< 0, if λ on a_i rarely co-occurs with λ' on a_j.

$= -1$, if λ on a_i never co-occurs with λ' on a_j.

$= 0$, if λ on a_i occurs independently of λ' on a_j.

With these compatibilities, the contextual knowledge is incorporated into the probabilistic relaxation operator.

The initial label likelihoods are provided from a classification of multispectral Landsat data. A multispectral scanner (MSS) gathers radiance

data in various sections of the electromagnetic spectrum (wavelength bands). For example, the Landsat MSS has four wavelength bands in the visible and near infrared spectrum. Such remotely sensed data have been collected and stored in digital format.

The first step in multispectral classification begins with the selection of an MSS data set which has sufficient quality so that the classes of interest on the land covered can be identified with the desired accuracy. After choosing the best available data set for analysis, the study area within the data is located and the reference data (such as aerial photography, maps, etc.) are correlated with the multispectral scanner data. These reference data provide the key to relating successfully the spectral responses in the data to the cover types on the ground.

The second step is the selection of the training samples. A common procedure for selecting the training areas is to use the available reference data to identify areas that contain the information classes of interest. The images of these areas are then identified in the multispectral scanner data. These training samples are used to determine parameters for the pattern recognition algorithms, effectively "training" the computer to recognize the classes of interest. Later when the classification operation is carried out by the pattern recognition algorithm, each data point to be classified is compared with the training samples for each class, and the pixel is assigned to the class it resembles most closely.

The third step is to use these training samples to define training classes. The training classes are often characterized in terms of the mean vectors and covariance matrices by the clustering algorithm. Clustering may be used to identify natural spectral groupings of pixels in the training samples. These

natural groupings, called "spectral classes," are used as candidate training classes. To be sure of the reliability in identifying the information class of each cluster class obtained, all available reference data are used. This step is the most important in ensuring that the classifier is correctly trained.

The final step is classification using a maximum likelihood classification algorithm. The set of discriminant functions for the maximum likelihood classification rule, usually specified in terms of mean vectors and covariance matrices of classes, is derived from the statistical decision theory so as to minimize the probability of making an erroneous classification. When the data values of a pixel are substituted into all of the functions, the pixel is assigned to the class which produces the largest value.

The initial likelihoods of the labelings for each pixel are provided by the values of the discriminant functions of the classification algorithm. Each probability is then updated by a rule of the form:

$$P_i^{(k+1)}(\lambda) = \frac{P_i^{(k)}(\lambda)\,[1 + q_i^{(k)}(\lambda)]}{\sum_{\lambda \in \Lambda} P_i^{(k)}(\lambda)\,[1 + q_i^{(k)}(\lambda)]} \tag{2.1.1}$$

where

$$q_i^{(k)}(\lambda) = \sum_{j \in J} d_{ij} \sum_{\lambda' \in \Lambda} \gamma_{ij}(\lambda,\lambda') P_j^{(k)}(\lambda') \tag{2.1.2}$$

where k is the iteration of the relaxation process and $q_i^{(k)}(\lambda)$ denotes the kth estimate of the neighborhood contribution. J defines the neighborhood about the current pixel being considered. The coefficients d_{ij} represent the possible weighting constants (which satisfy $\sum_{j \in J} d_{ij} = 1$) over the neighboring objects a_j. These coefficients insure that q_i is in the range [-1,1] and allow different

neighbors in J to have different degrees of influence in the neighborhood contribution. Indeed, if $P_j^{(k)}(\lambda')$ is high, and $\gamma_{ij}(\lambda,\lambda')$ is very positive or very negative, then the label λ' at a_j makes a substantial positive or negative contribution to $q_i^{(k)}(\lambda)$; while if $P_j^{(k)}(\lambda')$ is low, λ' at a_j makes relatively little contribution to $q_i^{(k)}(\lambda)$ regardless of the value of $\gamma_{ij}(\lambda,\lambda')$. Therefore, a very positive or very negative contribution to $q_i^{(k)}$ contributes an increase or decrease to $P_i^{(k)}(\lambda)$ since $P_i^{(k)}(\lambda)$ is obtained by multiplying $P_i^{(k)}$ by $(1 + q_i^{(k)})$, whereas a small contribution to $q_i^{(k)}$ contributes little change to $P_i^{(k+1)}$. Here, *the denominator in Equation* (2.1.1) guarantees that all the P's sum to 1. Moreover, they remain nonnegative, since $q_i^{(k)}$ is in the range [-1,1] provided that $\sum_j d_{ij} = 1$, so $1 + q_i^{(k)}$ is nonnegative.

This rule is used to update the likelihood of each label on each object in parallel, and is then iterated until no further changes occur. At this point, we say that the final labelings reach a balance in consistency between spectral and spatial (or contextual) data sources of information.

2.1.2 Compatibility Coefficients as Conditional Probabilities

The original iterative rule of Equation (2.1.1) uses a priori knowledge embedded in the d_{ij} and γ_{ij} functions to disambiguate the initial likelihoods. In the original design [3], the $\gamma_{ij}(\lambda,\lambda')$ coefficients were regarded as representing correlation functions. This association turns out to be in general inadequate. The representation of conditional likelihoods seems to be appropriate. Such conditional measures are in the form of "given λ' on a_j, how compatible is this with λ on a_i?". Now, we shall transform the updating rule of Equation (2.1.1) into one involving conditional likelihoods. The range of values for

compatibilities, [-1,1], must be scaled into [0,1]. The simplest transformation is:

$$P_{ij}(\lambda|\lambda') = \frac{\gamma_{ij}(\lambda,\lambda') + 1}{\alpha} \tag{2.1.3}$$

where $P_{ij}(\lambda|\lambda')$ is to be read as the conditional probability that a_i has label λ given a_j has λ'. α is a suitable constant. Substituting Equation (2.1.3) into Equation (2.1.1) gives:

$$P_i^{(k+1)}(\lambda) = \frac{P_i^{(k)}(\lambda)\{1 + \sum_j d_{ij} \sum_{\lambda'} [\alpha P_{ij}(\lambda|\lambda') - 1] P_j^{(k)}(\lambda')\}}{\sum_\lambda P_i^{(k)}(\lambda)\{1 + \sum_j d_{ij} \sum_{\lambda'} [\alpha P_{ij}(\lambda|\lambda') - 1] P_j^{(k)}(\lambda')\}}$$

$$= \frac{P_i^{(k)}(\lambda)\{1 + \sum_j d_{ij} \sum_{\lambda'} \alpha P_{ij}(\lambda|\lambda') P_j^{(k)}(\lambda') - 1\}}{\sum_\lambda P_i^{(k)}(\lambda)\{1 + \sum_j d_{ij} \sum_{\lambda'} \alpha P_{ij}(\lambda|\lambda') P_j^{(k)}(\lambda') - 1\}}$$

$$= \frac{P_i^{(k)}(\lambda)\{\sum_j d_{ij} \sum_{\lambda'} \alpha P_{ij}(\lambda|\lambda') P_j^{(k)}(\lambda')\}}{\sum_\lambda P_i^{(k)}(\lambda)\{\sum_j d_{ij} \sum_{\lambda'} \alpha P_{ij}(\lambda|\lambda') P_j^{(k)}(\lambda')\}}$$

$$= \frac{P_i^{(k)}(\lambda)\{\sum_j d_{ij} \sum_{\lambda'} P_{ij}(\lambda|\lambda') P_j^{(k)}(\lambda')\}}{\sum_\lambda P_i^{(k)}(\lambda)\{\sum_j d_{ij} \sum_{\lambda'} P_{ij}(\lambda|\lambda') P_j^{(k)}(\lambda')\}}$$

$$= \frac{P_i^{(k)}(\lambda) Q_i^{(k)}(\lambda)}{\sum_\lambda P_i^{(k)}(\lambda) Q_i^{(k)}(\lambda)} \tag{2.1.4}$$

where

$$Q_i^{(k)}(\lambda) = \sum_{j \in J} d_{ij} \sum_{\lambda' \in \Lambda} P_{ij}(\lambda \mid \lambda') P_j^{(k)}(\lambda') \qquad (2.1.5)$$

This is an analogous form of the updating rule of Equation (2.1.1) with compatibility coefficients replaced by conditional likelihoods which satisfy $\sum_{\lambda \in \Lambda} P_{ij}(\lambda \mid \lambda') = 1$.

2.1.3 Problem

The problem we want to deal with here is how to incorporate information from multiple ancillary data sources into the results of an existing classification of remotely sensed data. Conventional classification of cover types in remotely sensed data, based only upon spectral information, might not be enough. In many cases, there are other sources of practical data available which can be used along with the spectral information to improve the classification. For example, various tree species are known to exhibit growth patterns dependent upon topographic position, as shown in Fig. 2.1.1, so that tree species or other forest classifications could be improved by combining spectral data with elevation, slope, and other topographically related information. As shown in Fig. 2.1.1, the current pixel with remotely sensed data $[x_1, x_2, x_3, x_4]$ gathered from four wavelength bands from visible and near infrared spectrum has other ancillary data y_1, y_2, y_3 obtained from information of elevation, slope and aspect. The information from ancillary data sources is assumed to be available in the form of a set of likelihoods. The procedures proposed below are post-classification techniques, in that the influence of the available ancillary data can be imposed on an existing, spectrally determined, classification result.

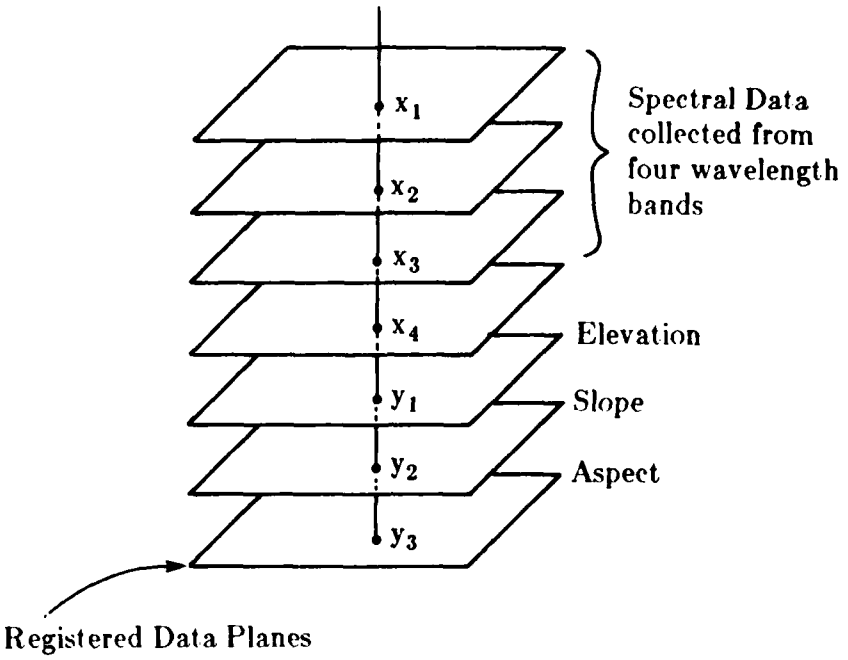


Fig. 2.1.1. Use of Ancillary Data [2]

2.1.4 Approach: Supervised Relaxation Labeling

A method of supervised relaxation labeling [12,2] has been proposed for incorporating one source of ancillary information into a classification in a quantitative manner. This method was previously applied only to a two-class (spruce-fir vs. others) problem. The supervised relaxation labeling can develop consistency between spectral, spatial, and multiple ancillary data sources of information for problems with multiple classes. The classification used for this purpose was produced from multispectral Skylab imagery. For example, distribution of the classes of tree species with respect to the elevation in one specific area is shown in Fig. 2.1.3 For simplicity, the area is assumed to be labeled into three classes, Spruce Fir, Douglas & White Fir, and Ponderosa Pine. But the method can be applied in a similar way to problems involving more classes. In this area, there are also data describing the elevation, slope, and aspect preferences of the various tree species, along with digitized terrain maps of elevation, slope, and aspect. For the present study, elevation was chosen as the most important ancillary data variable for improving classification accuracy over an existing classification obtained from spectral data alone. An elevation map of the area is shown in Fig. 2.1.2 [2] and the distributions of three classes (assume their labels are λ_1, λ_2, and λ_3) with respect to elevation are shown in Fig. 2.1.3. Given the elevation of the current pixel known from Fig 2.1.2 the probability of finding classes labeled λ_1, λ_2, and λ_3 can be seen in Fig. 2.1.3. Let the likelihoods be x, y, and z. Then a set of relative likelihoods (corresponding to each pixel in an image) for each of the labels for the current pixel a_i is as follows:

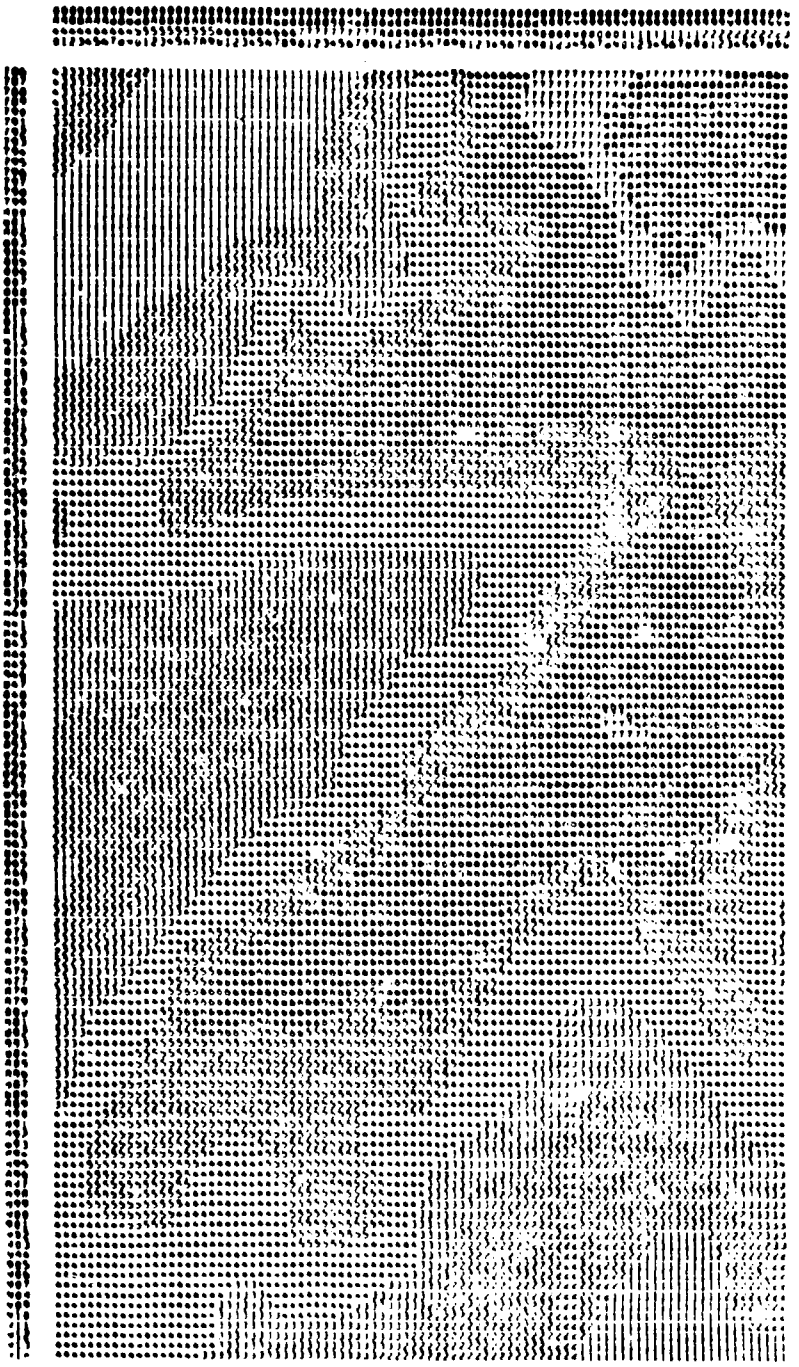


Fig. 2.1.2. Digitized Terrain Map [2]

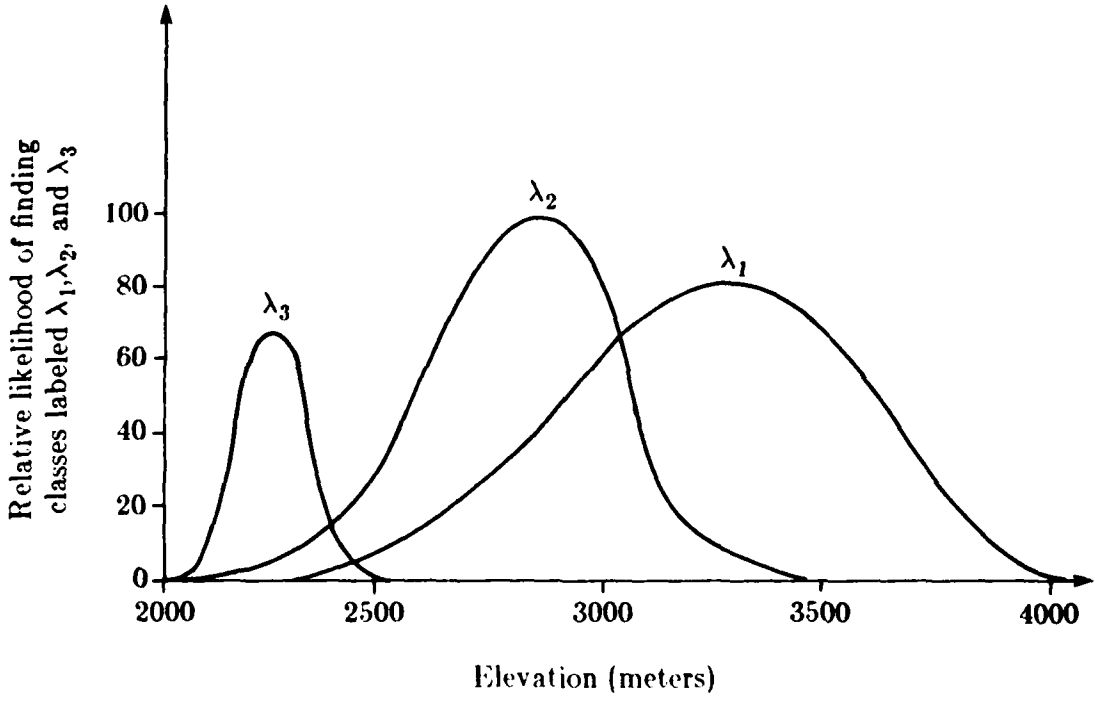


Fig. 2.1.3. Distribution Functions Illustrating the Relative Likelihood of Finding Three Classes at Various Altitudes [1]

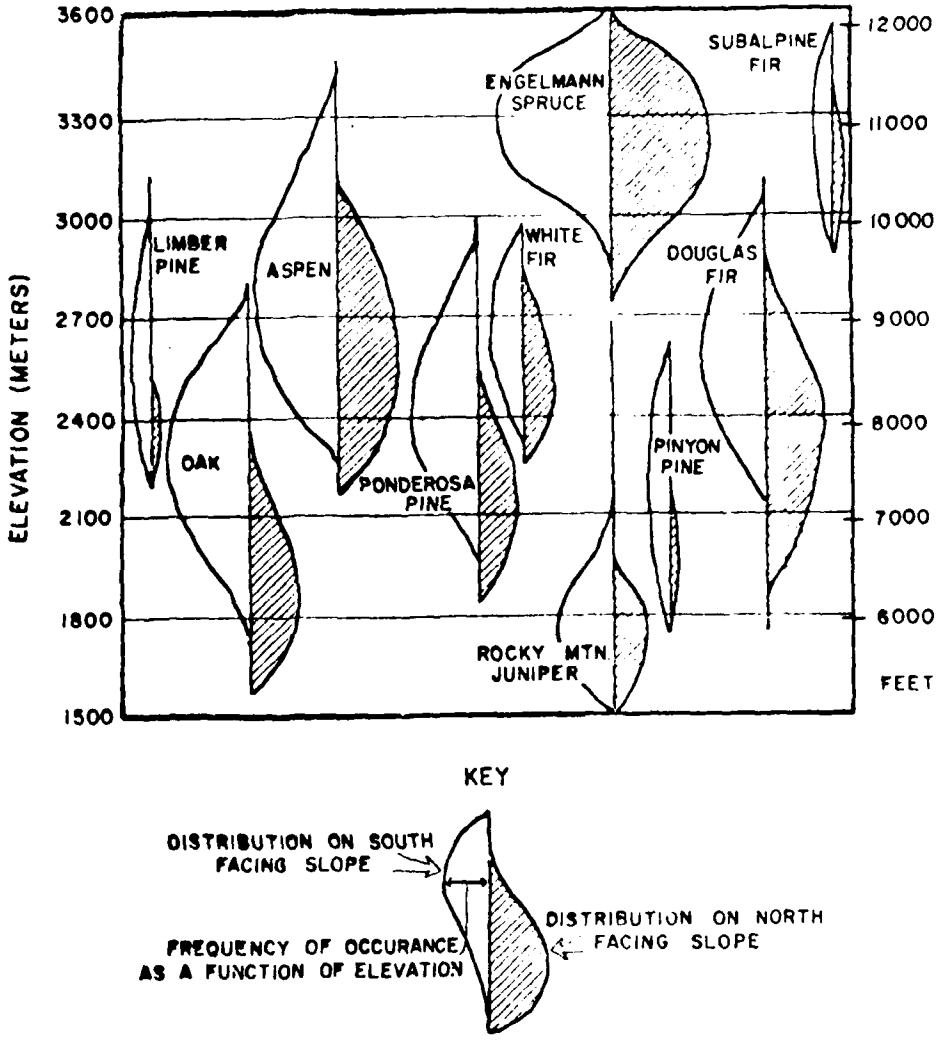


Fig. 2.1.4. Forest Species Distribution as a Function of Elevation and Aspect in the San Juan Mountains [1]

$$\phi_i(\lambda_1) = \frac{x}{x + y + z}$$

$$\phi_i(\lambda_2) = \frac{y}{x + y + z}$$

$$\phi_i(\lambda_3) = \frac{z}{x + y + z}$$

where

$$\sum_j \phi_i(\lambda_j) = 1$$

These likelihoods have been derived from the ancillary data (i.e., elevation) apart from the spectral data used in determining the $P_i(\lambda)$'s in Equation (2.1.1) and Equation (2.1.4).

If there are multiple sources of ancillary data available, how can we incorporate these multiple ancillary sources into the result of an existing classification? Let us denote the set of likelihoods derived from the first ancillary data source for pixel a_i as $\Phi_i^1 = [\phi_i^1(\lambda_1), \phi_i^1(\lambda_2), \phi_i^1(\lambda_3)]^T$ and that derived from the second ancillary data source for pixel a_i as $\Phi_i^2 = [\phi_i^2(\lambda_1), \phi_i^2(\lambda_2), \phi_i^2(\lambda_3)]^T$. Then the final set of relative strengths (that describe the relative likelihood of the labels for the pixel a_i) derived from these two ancillary data sources is denoted as $\Phi_i = [\phi_i(\lambda_1), \phi_i(\lambda_2), \phi_i(\lambda_3)]^T$ where Φ_i has the form of an elementwise product of two vectors Φ_i^1 and Φ_i^2

$$\Phi_i = \begin{bmatrix} \phi_i(\lambda_1) \\ \phi_i(\lambda_2) \\ \phi_i(\lambda_3) \end{bmatrix} = \begin{bmatrix} \dfrac{\phi_i^1(\lambda_1)\phi_i^2(\lambda_1)}{\phi_i^1(\lambda_1)\phi_i^2(\lambda_1)+\phi_i^1(\lambda_2)\phi_i^2(\lambda_2)+\phi_i^1 \text{l3pi2}(\lambda_3)} \\ \dfrac{\phi_i^1(\lambda_2)\phi_i^2(\lambda_2)}{\phi_i^1(\lambda_1)\phi_i^2(\lambda_1)+\phi_i^1(\lambda_2)\phi_i^2(\lambda_2)+\phi_i^1(\lambda_3)\phi_i^2(\lambda_3)} \\ \dfrac{\phi_i^1(\lambda_3)\phi_i^2(\lambda_3)}{\phi_i^1(\lambda_1)\phi_i^2(\lambda_1)+\phi_i^1(\lambda_2)\phi_i^2(\lambda_2)+\phi_i^1 \text{l3pi213}} \end{bmatrix}$$

where

$$\sum_{j=1}^{3} \phi_i(\lambda_j) = 1$$

The final set of relative strengths has been normalized to make the sum of all the terms equal 1. Of course, this approach can be generalized to deal with problems with multiple-class, multiple-ancillary data sources. Therefore, the modified supervised relaxation procedure is adopted to incorporate the $\phi_i(\lambda)$'s and thus allows multiple sources of ancillary data to bias the outcome of the set of likelihoods for each pixel in the image at each iteration. If the current favored label on the pixel is also strongly supported by the ancillary data [as expressed by $\phi_i(\lambda)$], then its likelihood is strengthened prior to moving to the next iteration. Conversely, if the current favored label is not strongly supported by the ancillary data, then its probability is weakened before proceeding. From the form of Φ_i, it is easy to realize the reason why an elementwise product rule is chosen to combine the sets of likelihoods derived from multiple ancillary sources. Namely, any label (or class name) with a very small value of probability derived from ancillary data source will force that label to have only very limited effects on the next estimate for the current pixel a_i even though other ancillary sources have evidence to support this label. It further prevents the additive accumulation of the small responses from

ancillary data sources due to some inaccurate estimations. The strategy of modified supervised relaxation labeling can be implemented quantitatively by defining

$$v_i(\lambda) = 1 + \beta[N\phi_i(\lambda) - 1], \quad \lambda \epsilon \Lambda \tag{2.1.6}$$

where

N is the total number of possible labels (i.e., classes)

β is a weighting constant to be determined

Through the parameter β, the present method allows the relative influence of spectral and ancillary data sources to be varied. A large β reflects a high degree of confidence in the accuracy of ancillary data sources. At the k+1 th iteration, Equation (2.1.6) is used to modify the label weights according to

$$P_i^{(k+1)}(\lambda)^+ = \frac{P_i^{(k+1)}(\lambda)v_i(\lambda)}{\sum_{\lambda \epsilon \Lambda} P_i^{(k+1)}(\lambda)v_i(\lambda)} \tag{2.1.7}$$

where the denominator is a normalizing factor. If $\beta = 0$, then $v_i(\lambda) = 1$ and $P_i^{(k+1)}(\lambda)^+ = P_i^{(k+1)}(\lambda)$, which means the ancillary data have no influence at all. If the ancillary data sources have provided no information (or no preference for any label for the pixel a_i), then $\phi_i(\lambda) = N^{-1}$ for $\lambda \epsilon \Lambda$. This leads to $P_i^{(k+1)}(\lambda)^+ = P_i^{(k+1)}(\lambda)$ i.e., the ancillary data sources have no influence on the progress of the relaxation.

Substituting Equation (2.1.4) into Equation (2.1.7), it becomes

$$P_i^{(k+1)}(\lambda)^+ = \frac{P_i^{(k)}(\lambda)Q_{i\Lambda}^{(k)}(\lambda)}{\sum_{\lambda \epsilon \Lambda} P_i^{(k)}(\lambda)Q_{i\Lambda}^{(k)}(\lambda)} \tag{2.1.8}$$

where

$$Q_{iA}^{(k)}(\lambda) = Q_i^{(k)}(\lambda) * \phi_i(\lambda) \qquad (2.1.9)$$

$Q_{iA}^{(k)}(\lambda)$ is the original neighborhood contribution of Equation (2.1.5) augmented by Equation (2.1.6).

The modified supervised relaxation labeling starts with $P_i^{(0)}(\lambda)$, $\lambda \in \Lambda$ where $P_i^{(0)}(\lambda)$ is the initial probability. In principle, these likelihoods are available in the penultimate step of the maximum likelihood classification obtained using spectral data only (i.e., just prior to the pixel being labeled according to the largest of those likelihoods).

The required context conditional likelihoods $P_{ij}(\lambda \mid \lambda')$'s are computed from the original classification. A more precise means for obtaining the compatibilities would be to estimate them from some other source of data known to be correct.

The final labeling achieved will represent a balance in consistency between the spectral information implicit in the initial likelihoods $P_i^{(0)}(\lambda)$, $\lambda \in \Lambda$, ancillary information embedded in the $\phi_i(\lambda)$'s, and spatial context data incorporated in the set of $P_{ij}(\lambda \mid \lambda')$'s.

2.2 Method of the Integration of Image Data from Multiple Ancillary Data Sources

In Section 2.1.4, an elementwise product of two vectors Φ_i^1 and Φ_i^2 which are derived from two ancillary data sources for a pixel a_i is used to produce the final set of relative strengths. This section will provide a justification for using the elementwise product of vectors to integrate ancillary information from multiple sources.

$\phi_i(\lambda_j)$, in Section 2.1.4, is actually the probability of finding class λ_j if the elevation value at pixel a_i is given. Indeed, $\phi_i(\lambda_j)$ can be expressed as $P(\lambda_j|\, x)$ in which index i is omitted and x is the elevation value. That is, given the elevation value x, the probability of finding class λ_j is $P(\lambda_j|\, x)$. Similarly, given another value y such as slope or aspect angle, another probability $P(\lambda_j|\, y)$ can be found. Then, the multiplication of these two values, $P(\lambda_j|\, x)P(\lambda_j|\, y)$, can be expressed as follows, if it is assumed that $P(x)P(y) = P(x,y)$ and $P(x|\, \lambda_j)P(y|\, \lambda_j) = P(x,y|\, \lambda_j)$:

$$
\begin{aligned}
P(\lambda_j|\, x)P(\lambda_j|\, y) &= \frac{P(x|\, \lambda_j)P(\lambda_j)}{P(x)} \cdot \frac{P(y|\, \lambda_j)P(\lambda_j)}{P(y)} \\
&= \frac{P(x,y|\, \lambda_j)P(\lambda_j)P(\lambda_j)}{P(x,y)} \\
&= \frac{P(x,y,\lambda_j)P(\lambda_j)}{P(x,y)} \\
&= P(\lambda_j|\, x,y)P(\lambda_j) \\
&= \frac{P(\lambda_j|\, x,y)}{N} \qquad (2.2.1)
\end{aligned}
$$

where, in the last step, $P(\lambda_j) = \frac{1}{N}$ and N is the total number of classes. This means that each class is assumed to be equally likely. After normalization as in Section 2.1.4, the constant term, N, in the above equation will be eliminated. Then, the probability of finding class λ_j given two ancillary data observations x and y can be derived from the probability of finding class λ_j given x and the probability of finding class λ_j given y. If the independence assumptions are not valid, the derivation in Equation (2.2.1) is not valid either. In this case, $P(\lambda_j|\, x,y)$ can be expressed as

$$P(\lambda_j | x,y) = \frac{P(x,y | \lambda_j)P(\lambda_j)}{P(x,y)} \qquad (2.2.2)$$

The joint distributions, $P(x,y | \lambda_j)$ must be estimated first through the training samples, and then the probability, $P(\lambda_j | x,y)$, can be calculated.

Given more than two independent ancillary data sources, the probability of finding class λ_j can be derived in exactly the same way as in Equation (2.2.1) except that the constant term will be different. Again, the constant term will be eliminated after normalization.

2.3 Convergence Property

Section 2.3.6 will discuss the convergence property of the supervised relaxation operator. The preceding material will describe some theoretical background, most closely following [13], step by step in order to reach the final conclusion in the last Section 2.3.6.

2.3.1 Introduction

In a relaxation operator, there are:

(1) a set of objects;

(2) a set of labels for each object;

(3) a neighbor relation over the objects; and

(4) a constraint relation over labels at pairs of neighboring objects

We shall denote the objects by the variable i, $i \in A$, which can take on integer values between 1 and n (the number of objects), the set of labels attached to node i by Λ_i, and the individual label (elements of Λ_i) by the variable λ.

For simplicity, we assume that the number of labels at each node is m, independent of i, so that the variable λ takes on integer values from 1 to m.

Constraints are only defined over neighboring nodes. The constraints allow for labels to express a preference or relative dislike for other labels at neighboring nodes by use of weighted values representing relative preferences. That is, the constraints are generalized to real-valued compatibility function $\gamma_{ij}(\lambda,\lambda')$ signifying the relative support for label λ at object i that arises from label λ' at object j. This support can be either positive or negative. Generally, positive values indicate that labels form a locally consistent pair, whereas a negative value indicates an implied inconsistency. When there is no interaction between labels, or when i and j are not neighbors, $\gamma_{ij}(\lambda,\lambda')$ is zero.

Having given the compatibility weights, continuous relaxation also uses weights for label assignments. We denote the weight with which label λ is assigned to node i by $P_i(\lambda)$, and will require that

$$0 \leq P_i(\lambda) \leq 1, \quad \text{all } i, \lambda$$

and

$$\sum_{\lambda=1}^{m} P_i(\lambda) = 1, \quad \text{all } i = 1,\ldots,n.$$

The relaxation process iteratively updates the weighted label assignments to be more consistent with neighboring labels, so that the weights designate a unique label at each node. Some have defined consistency as the stopping points of a standard relaxation labeling algorithm.

2.3.2 Consistency

An unambiguous labeling assignment is a mapping from the set of objects into the set of all labels, so that each object is associated with exactly one label.

$$P_i(\lambda) = \begin{cases} 1 & \text{if object i maps to label } \lambda \\ 0 & \text{if object i does not map to } \lambda \end{cases}$$

Note that for each object i,

$$\sum_{\lambda=1}^{m} P_i(\lambda) = 1.$$

The variables $P_i(1), \ldots, P_i(m)$ can be viewed as composing an m-vector $\bar{P}_i$, and the concatenation of the vectors $\bar{P}_1, \bar{P}_2, \ldots, \bar{P}_n$ can be viewed as forming an assignment vector $\bar{P} \in \mathbf{R}^{nm}$. The space of unambiguous labelings is defined by

$$\begin{aligned} \mathbf{K}^* = \{\bar{P} \in \mathbf{R}^{nm}:\ & \bar{P} = (\bar{P}_1, \ldots, \bar{P}_n); \\ & \bar{P}_i = (P_i(1), \ldots, P_i(m)) \in \mathbf{R}^m; \\ & P_i(\lambda) = 0 \text{ or } 1, \text{ all i, } \lambda; \\ & \sum_{\lambda=1}^{m} P_i(\lambda) = 1 \quad \text{for i} = 1,\ldots,n\}\ . \end{aligned}$$

A weighted labeling assignment can be defined by replacing the condition $P_i(\lambda) = 0$ or 1 by the condition $0 \leq P_i(\lambda) \leq 1$ for all i and λ. The space of weighted labeling assignments is defined by

$$\mathbf{K} = \{\bar{P} \in \mathbf{R}^{nm}:\ \bar{P} = (\bar{P}_1, \ldots, \bar{P}_n);$$

$$\bar{P}_i = (P_i(1),\ldots,P_i(m)) \in \mathbf{R}^m;$$

$$0 \le P_i(\lambda) \le 1 \text{ for all } i, \lambda;$$

$$\sum_{\lambda=1}^{m} P_i(\lambda) = 1 \quad \text{for } i = 1,\ldots,n\}$$

We note that $\mathbf{K}$ is simply the convex hull of $\mathbf{K}^*$. For example,

$$\bar{P} = (\bar{P}_1, \bar{P}_2)$$

$$= \sum_{e_1=1}^{3} P_1(e_1)P_2(1)\cdot(\bar{e}_{e_1},\bar{e}_1) + \sum_{e_1=1}^{3} P_1(e_1)P_2(2)\cdot(\bar{e}_{e_1},\bar{e}_2) + \sum_{e_1},\bar{e}_3)$$

$$= P_1(1)P_2(1)\cdot(\bar{e}_1,\bar{e}_1) + P_1(2)P_2(1)\cdot(\bar{e}_2,\bar{e}_1) + P_1(3)P_2(1)\cdot(\bar{e}_3,\bar{e}_1)$$

$$+ P_1(1)P_2(2)\cdot(\bar{e}_1,\bar{e}_2) + P_1(2)P_2(2)\cdot(\bar{e}_2,\bar{e}_2) + P_1(3)P_2(2)\cdot(\bar{e}_3,\bar{e}_2)$$

$$+ P_1(1)P_2(3)\cdot(\bar{e}_1,\bar{e}_3) + P_1(2)P_2(3)\cdot(\bar{e}_2,\bar{e}_3) + P_1(3)P_2(3)\cdot(\bar{e}_3,\bar{e}_3)$$

$$= \left\{\begin{bmatrix} P_1(1)P_2(1) \\ P_1(2)P_2(1) \\ P_1(3)P_2(1) \end{bmatrix}, \begin{bmatrix} (P_1(1)+P_1(2)+P_1(3))P_2(1) \\ 0 \\ 0 \end{bmatrix}\right\}$$

$$+ \left\{\begin{bmatrix} P_1(1)P_2(2) \\ P_1(2)P_2(2) \\ P_1(3)P_2(2) \end{bmatrix}, \begin{bmatrix} 0 \\ (P_1(1)+P_1(2)+P_1(3))P_2(2) \\ 0 \end{bmatrix}\right\}$$

$$+ \left[\begin{bmatrix} P_1(1)P_2(3) \\ P_1(2)P_2(3) \\ P_1(3)P_2(3) \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ (P_1(1)+P_1(2)+P_1(3))P_2(3) \end{bmatrix}\right]$$

$$= \left[\begin{bmatrix} P_1(1) \\ P_1(2) \\ P_1(3) \end{bmatrix}, \begin{bmatrix} P_2(1) \\ P_2(2) \\ P_2(3) \end{bmatrix}\right] .$$

We can generalize this as follows: let $\bar{e}_k$ denote the standard unit m-vector with a 1 in the kth component. Then any labeling assignment $\bar{P}$ in **K** can be expressed by

$$\bar{P} = \sum_{e_1=1}^{m} \sum_{e_2=1}^{m} , \ldots , \sum_{e_n=1}^{m} P_1(e_1)P_2(e_2) \cdots P_n(e_n)\cdot(\bar{e}_{e_1},\bar{e}_{e_2}, \ldots ,\bar{e}_{e_n}) .$$

Since each nm-vector $(\bar{e}_{e_1}, \ldots ,\bar{e}_{e_n})$ is in $\mathbf{K}^*$, the above sum can be interpreted as a convex combination of the elements of $\mathbf{K}^*$. Note that the sum over all of the coefficients is 1 i.e.,

$$\sum_{e_1=1}^{m} \cdots \sum_{e_n=1}^{m} P_1(e_1) \cdots P_n(e_n) = 1$$

Remember that no restrictions are placed on the magnitudes of the compatibilities, $\gamma_{ij}(\lambda,\lambda')$'s, and these will be represented by a matrix of real numbers - the compatibility matrix.

Definition 2.3.1: Let a matrix of compatibility and a labeling assignment (unambiguous or not) be given. We define the support for label λ at object i for the assignment $\bar{P}$ by

$$S_i(\lambda) = S_i(\lambda;\bar{P}) = \sum_{j=1}^{n} \sum_{\lambda'=1}^{m} \gamma_{ij}(\lambda,\lambda')\, P_j(\lambda')$$

More generally, the support values $S_i(\lambda)$ combine to give a support vector $\bar{S}$ that is a function of $\bar{P}$, i.e. $\bar{S} = \bar{S}(\bar{P})$.

Definition 2.3.2 (Define consistency for unambiguous labelings):

Let $\bar{P} \in \mathbf{K}^*$ be an unambiguous labeling. Suppose that $\lambda_1, \ldots, \lambda_n$ are the labels which are assigned to objects 1,...,n by the labeling $\bar{P}$. That is, $\bar{P} = (\bar{e}_{\lambda_1}, \bar{e}_{\lambda_2}, \ldots, \bar{e}_{\lambda_n})$. The unambiguous labeling $\bar{P}$ is consistent (in $\mathbf{K}^*$) providing.

$$\begin{aligned} S_1(\lambda_1;\bar{P}) &\geq S_1(\lambda;\bar{P}), \quad 1 \leq \lambda \leq m \\ &\vdots \\ S_n(\lambda_n;\bar{P}) &\geq S_n(\lambda;\bar{P}), \quad 1 \leq \lambda \leq m \end{aligned}$$

At a consistent unambiguous labeling, the support, at each object, for the assigned label is the maximum support at that object. The condition for consistency in $\mathbf{K}^*$ can be restated as follows:

$$\sum_{\lambda=1}^{m} P_i(\lambda) S_i(\lambda;\bar{P}) \geq \sum_{\lambda=1}^{m} v_i(\lambda) S_i(\lambda;\bar{P}), \quad i = 1,\ldots,n$$

for all unambiguous labelings $\bar{v} \in \mathbf{K}^*$.

Definition 2.3.3 (Consistency for weighted labeling assignments):

Let $\bar{P} \in \mathbf{K}$ be a weighted labeling assignment. The $\bar{P}$ is consistent (in $\mathbf{K}$) providing

$$\sum_{\lambda=1}^{m} P_i(\lambda)S_i(\lambda;\bar{P}) \geq \sum_{\lambda=1}^{m} v_i(\lambda)S_i(\lambda;\bar{P}), \quad i = 1,...,n$$

for all labelings $\bar{v} \in \mathbf{K}$.

Definition 2.3.4 (Strictly consistent):

Let $\bar{P} \in \mathbf{K}$. Then $\bar{P}$ is strictly consistent providing

$$\sum_{\lambda=1}^{m} P_i(\lambda)S_i(\lambda;\bar{P}) > \sum_{\sigma=1}^{m} v_i(\lambda)S_i(\lambda;\bar{P}), \quad i = 1,...,n$$

for all labeling assignments $\bar{v} \in \mathbf{K}$, $\bar{v} \neq \bar{P}$.

Theorem 2.3.1: A labeling $\bar{P} \in \mathbf{K}$ is consistent if and only if $\bar{P} \in \mathbf{K}$: $\sum_{i,\lambda,j,\lambda'} \gamma_{ij}(\lambda,\lambda')P_j(\lambda')[v_i(\lambda)-P_i(\lambda)] \leq 0$ for all $\bar{v} \in \mathbf{K}$.

Proof: See [13].

Average local consistency is defined as:

$$\mathbf{A}(\bar{P}) = \sum_{i=1}^{n} \sum_{\lambda} P_i(\lambda)S_i(\lambda)$$

The individual components $S_i(\lambda)$ depend on $\bar{P}$ which varies during the relaxation process, whereas consistency occurs when $\sum v_i(\lambda)S_i(\lambda;\bar{P})$ is maximized by $\bar{v} = \bar{P}$. That is the $S_i(\lambda)$ should be fixed during the maximization.

Remember that the labeling space $\mathbf{K}$ is the convex hull of the unambiguous labeling assignment space $\mathbf{K}^*$.

2.3.3 Geometrical Structure of Assignment Space

A simple example: there are two objects with three possible labels for each object. A labeling assignment *consists of six nonnegative numbers*:

$$\bar{P} = (\bar{P}_1,\bar{P}_2) = (P_1(1),P_1(2),P_1(3);\ P_2(1),P_2(2),P_2(3)) \text{ satisfying}$$

$$\sum_{\lambda=1}^{3} P_i(\lambda) = 1, \quad \text{for } i = 1,2$$

The locus of possible subvectors $\bar{P}_1$ in $\mathbf{R}^3$ is shown in Fig. 2.3.1. The vector $\bar{P} = (\bar{P}_1,\bar{P}_2)$ can be regarded as two points, each lying in a copy of the likelihood space shown in Fig. 2.3.1. Thus **K** can be identified with the set of all pairs of points in two copies of the triangular space. This can be generalized to the case with n objects each with m labels. Then **K** is more complicated. A weighted labeling assignment is a point in the assignment space **K**, and **K** is in turn the convex hull of the set of unambiguous labeling assignments $\mathbf{K}^*$. *An unambiguous assignment* is composed of points which lie at vertices of their respective surfaces.

The tangent space is a surface which when placed at the given point lies "tangent" to the entire surface. If $\bar{P}$ is a labeling assignment in **K**, and $\bar{v}$ is any other assignment in **K**, the difference vector $\bar{d} = \bar{v} - \bar{P}$ is shown in Fig. 2.3.2. As $\bar{v}$ roams around **K**, the set of all possible tangent directions at $\bar{P}$ is swept out. The set of all tangent vectors at $\bar{P}$ is therefore given by

$$T_{\bar{P}} = \{\bar{d}:\ \bar{d} = \alpha(\bar{v} - \bar{P}),\ \bar{v} \in \mathbf{K},\ \alpha \geq 0\}$$

Any tangent vector is composed of n subvectors so that $\bar{d} = (\bar{d}_1, \ldots, \bar{d}_n)$ and

$$\sum_{\lambda=1}^{m} d_i(\lambda) = \sum_{\lambda=1} \alpha\cdot(v_i(\lambda) - P_i(\lambda)) = \alpha\cdot(1-1) = 0.$$

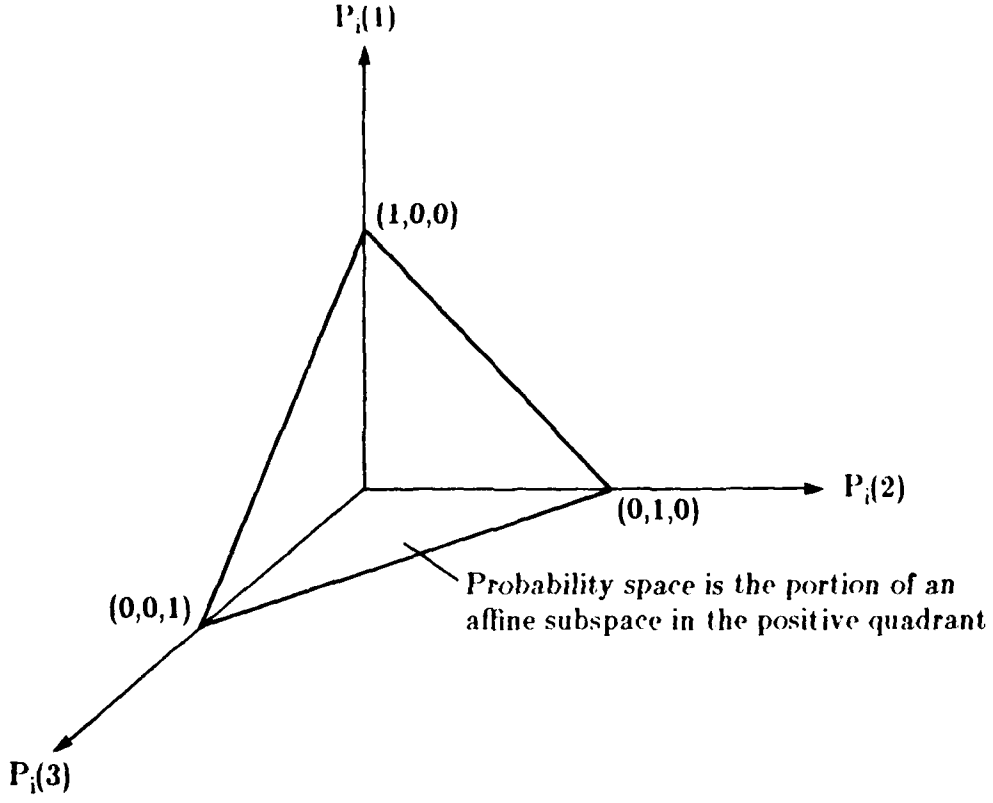


Fig. 2.3.1. Likelihood Space

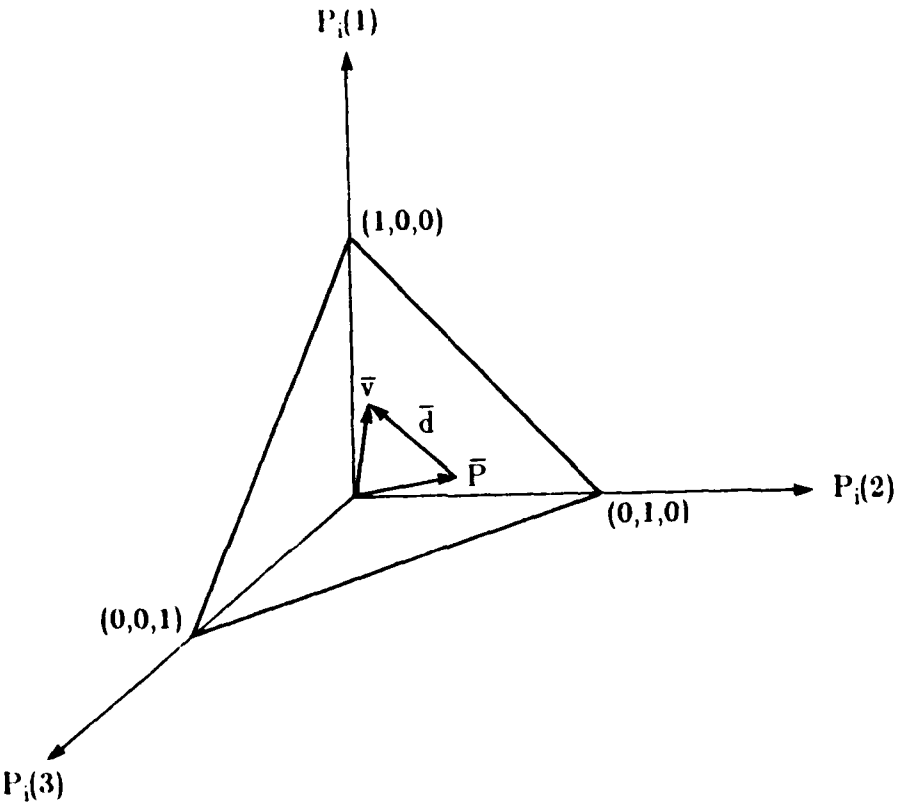


Fig. 2.3.2. Tangent Space

The set of tangent vectors at the interior point $\bar{P}$ consists of an entire subspace, which is given by

$$T_{\bar{P}} = \{\bar{d} = (\bar{d}_1, \ldots, \bar{d}_n);\ \bar{d}_i \in \mathbf{R}^m,\ \sum_{\lambda=1}^{m} d_i(\lambda) = 0\}$$

($\bar{P}$ interior to $\mathbf{K}$)

Observe that $T_{\bar{P}}$ and $\mathbf{K}$ are parallel flat surfaces.

When $\bar{P}$ lies on a boundary of $\mathbf{K}$, the tangent set is a proper subset of the above space $T_{\bar{P}_o}$ where $\bar{P}_o$ is any interior point. That is, when the assignment $\bar{P}$ has some zero components, the set of vectors of the form $\alpha\cdot(\bar{v}-\bar{P})$ is restricted to

$$T_{\bar{P}} = \{\bar{d} = (\bar{d}_1, \ldots, \bar{d}_n):\ \bar{d}_i \in \mathbf{R}^m,\ \sum_{\lambda=1}^{m} d_i(\lambda) = 0,$$

$$\text{and } d_i(\lambda) \geq 0 \text{ if } P_i(\lambda) = 0\}$$

2.3.4 *Maximizing Average Local Consistency*

From Theorem 2.3.1, maximizing $A(\bar{P})$ corresponds to finding a consistent labeling. The increase in $A(\bar{P})$ due to a small step of length α in the direction $\bar{u}$ is approximately the directional derivative:

$$A(\bar{P} + \alpha\bar{u}) - A(\bar{P}) \simeq \frac{d}{dt}\bigg|_{t=0} A(\bar{P} + t\alpha\bar{u}) = \text{grad } A(\bar{P})\cdot\alpha\bar{u}$$

where $\|u\| = 1$. The greatest increase in $A(\bar{P})$ can be expected if a step is taken in the tangent direction $\bar{u}$ which maximizes the directional derivative. However, if the directional derivative is negative or zero for all nonzero tangent directions, then $A(\bar{P})$ is a local maximum and no step should be taken. To find

a direction of steepest ascent, grad $A(\bar{P})\cdot\bar{u}$ should be maximized among the set of tangent vectors. However, it suffices to consider only those tangent vectors with Euclidean norm $\|\bar{u}\| = 1$ together with $\bar{u} = 0$.

Thus the direction of steepest ascent can be found by solving the following.

Problem 1: Find $\bar{u} \in T_{\bar{P}} \cap B_1(0)$ such that $\bar{u}\cdot\bar{q} \geq \bar{v}\cdot\bar{q}$ for all $\bar{v} \in T_{\bar{P}} \cap B_1(0)$, where $\bar{q} = \frac{1}{2}\,\text{grad}\,A(\bar{P})$. Here $B_1(0) = \{\bar{v} \in \mathbf{R}^{nm} : \|\bar{v}\| \leq 1\}$.

When the maximum $\bar{u}\cdot\bar{q} = 0$, we will agree that $\bar{u} = 0$ is the best solution to problem 1. Conceptually, starting at an initial labeling $\bar{P}$, we compute $\bar{q} = \frac{1}{2}\,\text{grad}\,A(\bar{P})$, and solve problem 1. If the resulting $\bar{u}$ is nonzero, we take a small step in the direction $\bar{u}$, and repeat the process. The algorithm terminates when $\bar{u} = 0$.

When $\bar{P}$ is the interior of the assignment space $\mathbf{K}$, solving problem 1 corresponds to projecting $\bar{q}$ onto the tangent space $T_{\bar{P}}$, and then normalizing.

Lemma 1: If $\bar{P}$ lies in the interior of $\mathbf{K}$, then the following algorithm solves problem 1.

(1) set $C_i = \frac{1}{m}\sum_{e=1}^{m} q_i(e)$, $i = 1,\ldots,n$.

(2) set $W_i(\lambda) = q_i(\lambda) - C_i$, all i, λ.

(3) set $u_i(\lambda) = W_i(\lambda)/\|\bar{W}\|$, all i, λ.

Here, $\|\bar{w}\| = \left[\sum_{i,\lambda} W_i(\lambda)^2\right]^{1/2}$

Proof:

Since $\|\bar{u}\| = 1$ and

$$\sum_{\lambda=1}^{m} u_i(\lambda) = \sum_{\lambda=1}^{m} (q_i(\lambda) - C_i)/\|\bar{w}\| = \left[\sum_{\lambda=1}^{m} q_i(\lambda) - mc_i\right]/\|\bar{w}\| = 0 \quad \text{for all } i$$

from the definition of tangent space, it is obvious that

$$\bar{u} \in T_{\bar{P}} \cap B_1(0)$$

To observe that $\bar{w}$ is the projection of $\bar{q}$ onto $T_{\bar{P}}$, we need to prove that $(\bar{q} - \bar{w}) \cdot \bar{v} = 0$ for all $\bar{v} \in T_{\bar{P}}$.

$$\sum_i \sum_\lambda (q_i(\lambda) - w_i(\lambda)) \cdot v_i(\lambda) = \sum_i c_i \sum_\lambda = 0 \quad \text{since } \bar{v} \in T_{\bar{P}}$$

Thus $\bar{v} \cdot \bar{q} = \bar{v} \cdot \bar{w}$ for all $\bar{v} \in T_{\bar{P}}$. Since $\bar{u} \in T_{\bar{P}}$, $\bar{u} \cdot \bar{q} = \bar{u} \cdot \bar{w} = \dfrac{\bar{w}}{\|\bar{w}\|} \cdot \bar{w} = \|\bar{w}\| \geq \|\bar{w}\| \cdot \|\bar{v}\|$ for any $\bar{v} \in T_{\bar{P}} \cap B_1(0)$ (note that $\|\bar{v}\| \leq 1$). Since $\|\bar{w}\| \cdot \|\bar{v}\| \geq \bar{w} \cdot \bar{v}$, so we have

$$\bar{u} \cdot \bar{q} \geq \bar{u} \cdot \bar{q} \quad \text{for all } \bar{v} \in T_{\bar{P}} \cap B_1(0)$$

That is, $\bar{u}$ solves problem 1.

Combining these results, we obtain the following algorithm for finding a local maximum of $\Lambda(\bar{P})$.

Algorithm 2.3.1:

Initialize:

(1) Start with an initial labeling assignment $\bar{P}^o \epsilon \mathbf{K}$. Set $k = 0$. Loop until a stop is executed:

(2) Compute $\bar{q}^k = \frac{1}{2} \text{grad} A(\bar{P}^k)$.

(3) Use the algorithm in Lemma 1, with $\bar{P} = \bar{P}^k$, $\bar{q} = \bar{q}^k$, to find the solution $\bar{u}^k$ to problem 1.

(4) If $\bar{u}^k = 0$ stop.

(5) Set $\bar{P}^{k+1} = \bar{P}^k + k\bar{u}^k$, where $0 < h \leq \alpha_k$ is determined so that $\bar{P}^{k+1} \epsilon \mathbf{K}$. The maximum step size α_k is some predetermined small value, and may decrease as k increases to facilitate convergence.

(6) Replace k by k + 1.

End loop.

In summary, successive iterates are obtained by moving a small step in the direction of the projection of the gradient $\bar{q}$ onto the convex set of tangent directions $T_{\bar{P}}$. The algorithm stops when this projection is zero. We now have a method for finding consistent labelings, given an initial labeling assignment. Recall the variational inequality for consistency from Theorem 2.3.1:

$$\sum_{i,\lambda} \sum_{j,\lambda'} \gamma_{ij}(\lambda,\lambda') P_j(\lambda')(v_i(\lambda) - P_i(\lambda)) \leq 0 \quad \text{for all } \bar{v} \epsilon \mathbf{K}$$

or, more generally,

$$\sum_{i,\lambda} S_i(\lambda;\bar{P}) \cdot (v_i(\lambda) - P_i(\lambda)) \leq 0 \text{ for all } \bar{v} \in \mathbf{K}$$

Hereafter, we define the components of $\bar{q}$ by

$$q_i(\lambda) = \sum_{j,\lambda'} \gamma_{ij}(\lambda,\lambda') P_j(\lambda') ,$$

that is, we have set $\bar{q} = \bar{S}(\bar{P})$.

Observation 2.3.1: With $\bar{q}$ defined as above, the variational inequality is equivalent to the statement

$$\bar{q} \cdot \bar{t} \leq 0 \text{ for all } \bar{t} \in T_{\bar{P}} .$$

That is, a labeling $\bar{P}$ is consistent if and only if $\bar{q}$ points away from all tangent directions.

Proof: We have $\bar{q} = \bar{S}$, and any tangent vector $\bar{t}$ at $\bar{P}$ can be written as a positive scalar multiple of $\bar{v} - \bar{P}$, where $\bar{v} \in \mathbf{K}$. The observation follows immediately.

Therefore, if at a labeling $\bar{P}$, the associated vector $\bar{q}$ points in the same direction as some tangent vector, then $\bar{P}$ is not consistent. So $\bar{P}$ should be moved in the direction of that tangent vector. The process may be repeated until $\bar{q}$ evaluated at the current assignment points away from all tangent directions. Then $\bar{P}$ will be a consistent labeling. Note that $\bar{q}$ varies as $\bar{P}$ moves, but that generally $\bar{q}$ will change smoothly and gradually.

If $\bar{q} \cdot \bar{t} > 0$ for some tangent direction $\bar{t}$, then the current assignment $\bar{P}$ is *not* consistent, and should be updated. It makes sense to move $\bar{P}$ in the direction $\bar{u}$ that maximizes $\bar{q} \cdot \bar{u}$. Therefore the relaxation labeling algorithm is given by the following.

Algorithm 2.3.2: Replace step 2 in Algorithm 2.3.1 with:

(2′) Compute $\bar{q} = \bar{S}(\bar{P})$. That is:

$$q_i(\lambda) = \sum_{j,\lambda'} \gamma_{ij}(\lambda,\lambda')\, P_j(\lambda')$$

All other steps remain the same.

Proposition 2.3.3: Suppose $\bar{P}$ is a stopping point of Algorithm 2.3.2. Then $\bar{P}$ is consistent.

Proof: A point $\bar{P}$ is a stopping point of Algorithm 2.3.2 if and only if $\bar{u} = 0$ solves problem 1. If $\bar{u} = 0$, then $\bar{v} \cdot \bar{q} \leq \bar{u} \cdot \bar{q} = \bar{0} \cdot \bar{q} = 0$ for all tangent vectors $\bar{v} \in T_{\bar{P}}$. On the other hand, if $\bar{t} \cdot \bar{q} \leq 0$ for all $\bar{t} \in T_{\bar{P}}$, then $\bar{u} = 0$ maximizes $\bar{u} \cdot \bar{q}$ for $\bar{u} \in T_{\bar{P}} \cap B_1(0)$. According to Observation 2.3.1, $\bar{t} \cdot \bar{q} \leq 0$ for all $\bar{t} \in T_{\bar{P}}$ is equivalent to the variational inequality, which is in turn equivalent to $\bar{P}$ being consistent (Theorem 2.3.1).

At this point, we have presented the relaxation labeling algorithm in such a way that the stopping points of the algorithm are consistent labelings.

Recall that a labeling is strictly consistent if

$$\sum_{\lambda} P_i(\lambda) S_i(\lambda) > \sum_{\lambda} v_i(\lambda) S_i(\lambda), \quad i = 1,\ldots,n$$

whenever $\bar{v} \neq \bar{P}$, $\bar{v} \in \mathbf{K}$. As a result, the variational inequality can be replaced by the statement

$$\sum_{i,\lambda,j,\lambda'} \gamma_{ij}(\lambda,\lambda') P_j(\lambda') (v_i(\lambda) - P_i(\lambda)) < 0$$

$$\text{for all } \bar{v} \in \mathbf{K}, \quad \bar{v} \neq \bar{P}$$

for a strictly consistent labeling. In particular, $\bar{q} \cdot \bar{u} < 0$ for all nonzero tangent directions $\bar{u}$ at a strictly consistent labeling $\bar{P}$. We claim that $\bar{P} \in \mathbf{K}^*$ (i.e., that $\bar{P}$ is an unambiguous labeling). Suppose, for contradiction, that $0 < P_{i_o}(\lambda_o) < 1$ for some (i_o, λ_o). Then for some other λ_o', $0 < P_{i_o}(\lambda_o') < 1$. We consider two tangent directions,

$$u_1(i,\lambda) = \begin{cases} 0 & , \quad \neq i_o \\ (0,\ldots,0,1,\ldots,-1,\ldots,0), & = i_o, \end{cases}$$

and $\bar{u}_2 = -\bar{u}_1$.

That is, $\bar{u}_1$ has a 1 in the (i_o,λ_o) position and a -1 in the (i_o,λ_o') position, and $\bar{u}_2$ is the other way around. These are valid tangent directions according to the formulation of $T_{\bar{P}}$. However, $\bar{q} \cdot \bar{u}_1 = -\bar{q} \cdot \bar{u}_2$, so they cannot both be negative. Hence, we have shown that a strictly consistent labeling $\bar{P}$ must be unambiguous. Thus if $\bar{q}$ points away from the surface $\mathbf{K}$ at a vertex (i.e., an unambiguous consistent labeling), then $\bar{q}$ will point generally toward the vertex at nearby assignments in $\mathbf{K}$. Accordingly, if $\bar{P}$ is near the unambiguous consistent labeling, moving $\bar{P}$ in a tangent direction $\bar{u}$ that points in the same direction as $\bar{q}$, should cause $\bar{P}$ to converge to the vertex.

2.3.5 Relaxation Operator

Algorithm 2.3.2 updates weighted labeling assignments by computing an intermediate vector $\bar{q}$, where

$$q_i(\lambda) = \sum_j \sum_{\lambda'} \gamma_{ij}(\lambda,\lambda') P_j(\lambda')$$

and then updating $\bar{P}$ in the direction defined by the projection of $\bar{q}$ onto $T_{\bar{P}}$.

As we shall show, the original updating formula [1] has the intermediate vector $\bar{q}$ defined by

$$q_i(\lambda) = \sum_j d_{ij} \sum_{\lambda'} \gamma_{ij}(\lambda,\lambda') P_j(\lambda')$$

In Algorithm 2.3.2, we set $q_i(\lambda) = S_i(\lambda)$. Here the support vector $\bar{S}$ is a *function* $\bar{S}(\bar{P})$, where $S_i(\lambda)$ is computed from a nonlinear function of current assignment values in $\bar{P}$. Presumably, $S_i(\lambda)$ depends on the components of $\bar{P}_j$ for objects j near object i, and is relatively independent of the values of $\bar{P}_j$ for objects distant from i. Therefore, the different object j near object i is given an influence on the support function $q_i(\lambda)$ which depends on the value of weighting constant d_{ij} and has no influence on $q_i(\lambda)$ at all if d_{ij} is zero.

The principle difference lies in the manner in which $\bar{q}$ is projected onto a tangent vector. In Algorithm 2.3.2, the tangent direction was obtained by maximizing $\bar{u} \cdot \bar{q}$ among $\bar{u} \in T_{\bar{P}} \cap B_1(0)$.

One of the standard relaxation formulas was suggested by Rosenfeld et al. [3] and is given by

$$P_i(\lambda) = \frac{P_i(\lambda)[1 + q_i(\lambda)]}{\sum_{e=1}^{m} P_i(e)[1 + q_i(e)]}$$

(It is assumed, when using this formula, that the $\gamma_{ij}(\lambda,\lambda')$ values are sufficiently small so that one can be sure that $|q_i(\lambda)| < 1$.) There is another similar formula, actually derived from this one by transforming compatibility coefficient $\gamma_{ij}(\lambda,\lambda')$ to conditional probability $P_{ij}(\lambda|\lambda')$. Therefore the proof of convergence for one is automatically valid for the other.

To consider the behavior of this standard formula, first assume that $\bar{P}$ is near the center of the assignment space, so that very approximately $P_i(\lambda) \simeq 1/m$ for all i, λ. The updating can then be regarded as consisting of two steps. First, the vector $\bar{P}$ is changed into an intermediate $\hat{P}$, where

$$\hat{P}_i(\lambda) = P_i(\lambda)\,[1 + q_i(\lambda)] \simeq P_i(\lambda) + q_i(\lambda)/m.$$

Next, $\hat{P}$ is normalized using a scalar constant for each object $\hat{P}_i$. When $\bar{P}$ is near the center of **K**, this rescaling process shifts $\hat{P}$ in a direction essentially perpendicular to **K**. That is, $\bar{P}$ is reset to approximately the projection of $\hat{P}$ onto **K**. Denoting the orthogonal projection operator by O_k, we have

$$\bar{P} = \bar{P}' \simeq O_k(\hat{P}) \simeq O_k(\bar{P} + \bar{q}/m)$$

by virtue of the continuity of O_k. Further, assuming that $\bar{P}$ is in the interior of **K**, and $\bar{q}$ is sufficiently small, then

$$O_k(\bar{P} + \bar{q}/m) = \bar{P} + \frac{1}{m}\, O_T(\bar{q})$$

where O_T is the orthogonal projection onto the linear subspace $T_{\bar{P}}$. However, the solution $\bar{u}$ to problem 1 is obtained by normalizing $O_T(\bar{q})$. Combining, we have that

$$\bar{P} \simeq \bar{P}' + \alpha\,\bar{u}$$

for some scalar α. Thus, $\bar{P}$ is reset to a vector which is approximately the updated vector that one would obtain by Algorithm 2.3.2.

When $\bar{P}$ is close to an edge or corner, the situation is somewhat more complicated. The first step in standard updating (i.e., $\hat{P}_i(\lambda) = P_i(\lambda)[1 + q_i(\lambda)] = P_i(\lambda) + P_i(\lambda)q_i(\lambda)$) can be viewed as an initial operation changing $\bar{q}$, since the components of $\bar{q}$ corresponding to small

components of $\bar{P}$ have minimal effect (i.e., the motions in directions perpendicular to the nearby edges are scaled down). The normalization step is the same as before. Therefore, the formula results in attenuation of motion perpendicular to an edge. Further, a zero component can never become nonzero even if the evidence supports the value.

2.3.6 Supervised Relaxation Operator

Now we are prepared to establish the convergence property of the supervised relaxation operator.

The original relaxation operator [3] indicates that $P_i(\lambda)$ is updated by $P_i(\lambda)[1 + q_i(\lambda)]$ in which $q_i(\lambda)$ is the neighborhood function. Actually, $P_i(\lambda)[1 + q_i(\lambda)] = P_i(\lambda) + P_i(\lambda)q_i(\lambda)$. These m likelihoods, $P_i(\lambda)$, $\lambda = 1,...,m$, form a m-vector, $\bar{P}_i$, for the current object i. Similarly, $P_i(\lambda)q_i(\lambda)$, $\lambda = 1,...,m$, form another m-vector, $\overline{P_i q_i}$. The formula, $P_i(\lambda)q_i(\lambda)$, implies that the neighborhood contribution to the class λ at current object i should remain relatively unchanged if the class λ at current object i has likelihood close to 1; otherwise it should decrease. The likelihood space of $\bar{P}_i$ and the vector $\overline{P_i q_i}$ are shown in Fig. 2.3.3, in which $\overline{P_i q_i}$ influences the movement direction of $\bar{P}_i$. After normalization, the summation of the components in the newly updated vector $\bar{P}_i'$ is equal to 1. That means the vector $\bar{P}_i + \overline{P_i q_i}$ is rescaled to make the new vector $\bar{P}_i'$ still in the likelihood space.

In the supervised relaxation algorithm, before $\bar{v}_i$, defined later, influences the movement direction of $\bar{P}_i$, the $P_i(\lambda)$ is influenced by $Q_i(\lambda)$ using the formula $P_i(\lambda)Q_i(\lambda)$ in which $Q_i(\lambda)$ is also a neighborhood contribution but the compatability coefficient $\gamma_{ij}(\lambda,\lambda')$ is expressed in terms of the conditional

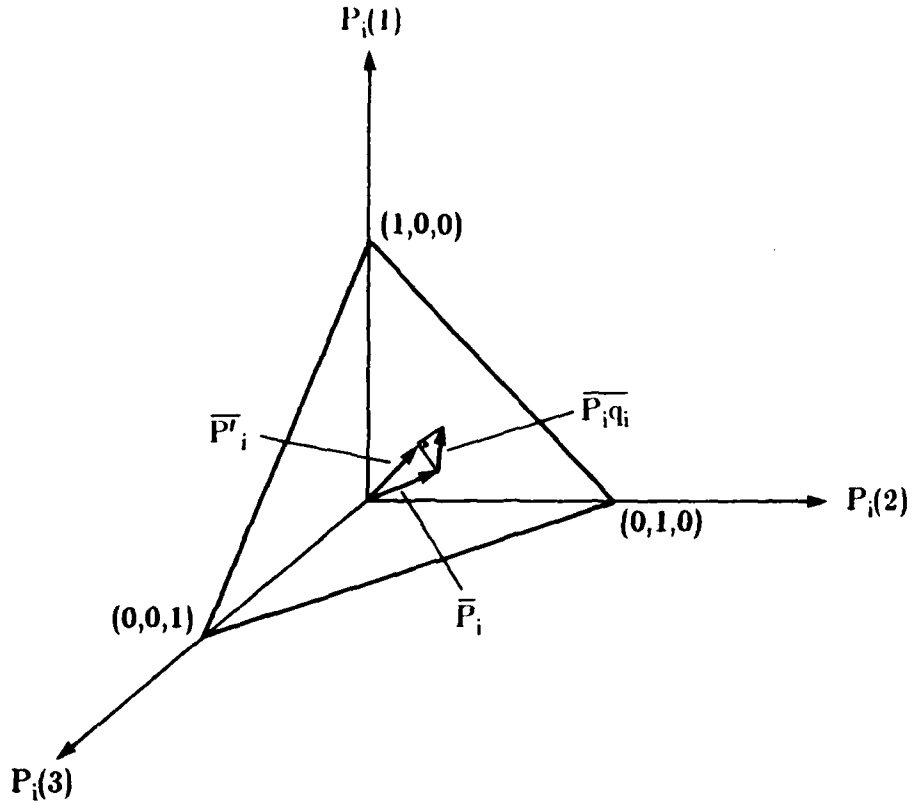


Fig. 2.3.3. Likelihood Space of Relaxation Operator

probability $P_{ij}(\lambda \mid \lambda')$. This can be seen in Section 2.1. $Q_i(\lambda)$, $\lambda = 1,...,m$, forms a m-vector $\bar{Q}_i$. For the supervised relaxation operator, the vector $\bar{P}_i'$ is then modified based on the probability derived from ancillary data sources using the formula $P_i^+(\lambda) = P_i'(\lambda)\psi_i(\lambda)$. Again, $\psi_i(\lambda)$, $\lambda = 1,...,m$, forms a m-vector $\bar{\psi}_i$. The vector $\bar{\psi}_i$ for every object i, $i = 1,...,n$, is fixed and doesn't change its component values while the supervised relaxation algorithm proceeds. But the vector $\bar{\psi}_i$ at an object i and $\bar{\psi}_j$ at an object j are likely to be different. That is, every object i has different vector $\bar{\psi}_i$. These three vectors, $\bar{P}_i, \bar{Q}_i$, and $\bar{\psi}_i$, are shown in Fig. 2.3.4 in which vectors $\bar{Q}_i$ and $\bar{\psi}_i$ compete with each other to influence the movement direction of $\bar{P}_i$. Of course, $\bar{P}_i$ has its preference to one of labels assigned to object i through the initial likelihoods $\overline{P_i^{(0)}}$. For example, if $\bar{\psi}_i$ and $\bar{Q}_i$ don't have enough influence to force $\bar{P}_i$ move away from the vectex (1,0,0), in which λ_1 is favored by $\overline{P_i^{(0)}}$, shown in Fig. 2.3 4, the final labeling of $\bar{P}_i$ will move toward and stay at this vertex (1,0,0); otherwise it will move to one of the other two vertices. From theorem 9.1 in [13], $\bar{P}_i$ will reach an unambiguous labeling assignment (i.e., moves to one of the three vertices) if $\bar{P}_i$ approaches sufficiently close to any one of these three vertices.

2.4 Simulation and Experimental Results

The *supervised relaxation* algorithm was programmed and applied to the analysis of a set of Landsat multispectral data. The data were collected by the satellite over the San Juan Mountains in SW Colorado [1]. The objective of the analysis was to discriminate among the ground cover classes such as "oak", "ponderosa pine", "aspen", "pasture", "douglas and white fir", "snow", "water", and "other", where the last category was simply a catchall. Each class was actually decomposed in the analysis process of clustering and merging

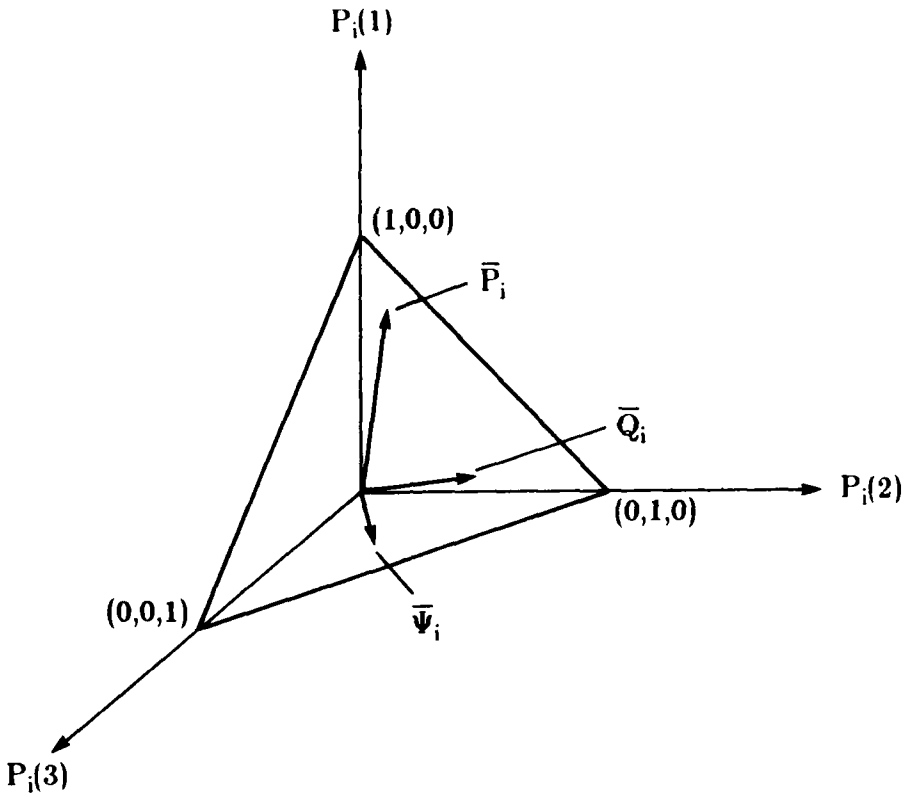


Fig. 2.3.4 Likelihood Space of Supervised Relaxation Operator

into a union of subclasses, each having a data distribution describable as approximately multivariate normal.

2.4.1 Results from the Maximum Likelihood Classification Algorithm

To provide a baseline for comparison, the data from the first and the second channels, which were in the range of visible wavelengths, were first analyzed using the maximum likelihood classification algorithm. The a priori likelihoods of the classes were approximated as being equal, and 2122 test samples, independent of the training samples, were used to evaluate the results. Due to different numbers of test samples in each class being used in the evaluation of the algorithm performance, the measure called average performance by class is used to avoid any bias toward any class which has the largest number of test samples. As shown in Table 2.4.1, the average performance by class of this conventional maximum likelihood classifier was 38.5 percent correct.

2.4.2 Results from the Relaxation Operator

To implement the relaxation analysis, the most formidable job is estimating both the initial likelihoods $P_i^{(0)}(\lambda)$, $\lambda \in \Lambda$ and the conditional likelihoods $P_{ij}(\lambda \mid \lambda')$. The initial likelihoods are available in the penultimate step of the maximum likelihood classification using 2-channel spectral data only (i.e., just prior to the pixel being labeled according to the largest of these likelihoods). The required context conditional likelihoods $P_{ij}(\lambda \mid \lambda')$ were estimated from the final classification results produced from the maximum likelihood classification algorithm by counting joint and individual occurrences of the classes. However, rather than computing four different sets of these

Table 2.4.1. Test Results for Classification of the Vallecito Quadrangle

	SAMP.	ASPEN	PPINE	DWF	PASTU	OAK	OTHERS
ASPEN	324	0	132	36	32	49	75
PPINE	664	0	388	98	75	74	29
DWF	996	0	195	644	100	10	47
PASTU	120	0	31	16	43	12	18
OAK	18	0	7	0	1	6	1

Ave. Performance by Class = 38.5

corresponding to each different neighbor type (left, right, above, and below), a single set was calculated by counting joint occurrences in both directions both vertically and horizontally.

The same test samples were used to evaluate the results. The results of this relaxation operator at iteration number 5, 10, and 20 are shown in Table 2.4.2. The final result was slightly better than the maximum likelihood classification. The average performance by class was 40.3 percent correct.

2.4.3 Results from the Supervised Relaxation Operator with One Ancillary Information Source

Due to the nonavailability of multiple ancillary information sources and the desire to demonstrate the feasibility of the algorithm, channel 3 data was used as ancillary data in the experiment. The supervised relaxation operator is now shown, by example, to be a useful tool for incorporating information from one ancillary data source, channel 3 data, into an existing classification produced from the maximum likelihood algorithm using 2-channel spectral data. Channel 3 is an infrared spectral band. The mean and standard deviation of each class having a data distribution describable as approximately normal were estimated for channel 3 using clustering and merge-statistics algorithm. From these data, sets of $\phi_i(\lambda)$ for each pixel in the image were generated and incorporated in the supervised relaxation algorithm. The same initial likelihoods and conditional likelihoods were again used. Several relaxation tests were performed using differing degrees of supervision, i.e., various weighting constants given to the influence of the ancillary data via the parameter β. The value of β which produced the best results was chosen.

Table 2.4.2. Test Results of Relaxation Operator at Iteration Number 5, 10, and 20

	SAMP.	ASPEN	PPINE	DWF	PASTU	OAK	OTHERS
ASPEN	324	0	207	41	1	7	68
PPINE	664	0	525	93	26	14	6
DWF	996	0	150	779	45	0	22
PASTU	120	0	45	18	43	0	14
OAK	18	0	10	0	1	0	7

Ave. Performance by Class = 38.6

(a) Results at Iteration 5

	SAMP.	ASPEN	PPINE	DWF	PASTU	OAK	OTHERS
ASPEN	324	0	220	41	0	4	59
PPINE	664	0	540	90	23	7	4
DWF	996	0	149	794	34	0	19
PASTU	120	0	45	20	43	0	12
OAK	18	0	11	0	1	0	6

Ave. Performance By Class = 39.4

(b) Results at Iteration 10

	SAMP.	ASPEN	PPINE	DWF	PASTU	OAK	OTHERS
ASPEN	324	0	234	41	0	0	49
PPINE	664	0	554	87	21	2	0
DWF	996	0	146	810	28	0	12
PASTU	120	0	46	20	44	0	10
OAK	18	0	11	0	1	0	6

Ave. Performance By Class = 40.3

(c) Results at Iteration 20

Again the same test samples were used to evaluate the results. As shown in Table 2.4.3, the average performance by class was better than the ordinary relaxation analysis. The final result is 64.1 percent correct. In addition, a closer look at the class-by-class results reveals that the performance for each class was better than those attained using the relaxation operator without ancillary information.

2.4.4 Results from the Supervised Relaxation Operator with Two Ancillary Information Sources

In this section, the supervised relaxation algorithm is shown to be an effective technique for incorporating information from two ancillary data sources, channel 3 data and elevation data, into an existing classification to see any improvement in classification accuracy over that obtained with only one ancillary source. Figure 2.1.4 shows the distribution of tree species as a function of elevation for an area northeast of the Vallecito Reservoir in the Colorado Rockies. Fig. 2.1.2 shows a digitized terrain map for the area covered by the multispectral scanner data described earlier. From these data, the second set of $\phi_i(\lambda)$ for each pixel in the image was generated and used along with those generated from the first set of $\phi_i(\lambda)$ in the supervised relaxation algorithm. The same initial likelihoods, conditional likelihoods, and test samples were used. The weight constant producing the best results was chosen. As shown in Table 2.4.4, the average performance by class using two ancillary data sources of information gave the best result, 80.8 percent accuracy. The results from the maximum likelihood algorithm, the relaxation algorithm and the supervised relaxation algorithm are compared and drawn in Fig. 2.4.1. From this figure, it is clear that the more information we use, the more

Table 2.4.3. Test Results of the Supervised Relaxation Algorithm with One Ancillary Information Source at Iteration Number 5, 10, and 20

	SAMP.	ASPEN	PPINE	DWF	PASTU	OAK	OTHERS
ASPEN	324	29	205	39	1	20	30
PPINE	664	0	568	58	20	16	2
DWF	996	3	114	865	1	0	13
PASTU	120	0	24	3	83	7	3
OAK	18	0	3	0	2	11	2

Ave. Performance By Class = 62.3

(a) Results at Iteration 5

	SAMP.	ASPEN	PPINE	DWF	PASTU	OAK	OTHERS
ASPEN	324	29	222	41	1	5	26
PPINE	664	0	579	54	18	11	2
DWF	996	0	98	890	0	0	8
PASTU	120	0	23	3	86	8	0
OAK	18	0	3	0	2	11	2

Ave. Performance By Class = 63.7

(b) Results at Iteration 10

	SAMP.	ASPEN	PPINE	DWF	PASTU	OAK	OTHERS
ASPEN	324	29	224	45	1	2	23
PPINE	664	0	586	52	18	8	0
DWF	996	0	95	893	0	0	8
PASTU	120	0	23	3	87	7	0
OAK	18	0	3	0	2	11	2

Ave. Performance By Class = 64.1

(c) Results at Iteration 20

Table 2.4.4. Test Results of the Supervised Relaxation Algorithm with Two Ancillary Information Sources at Iteration Number 5, 10, and 20

	SAMP.	ASPEN	PPINE	DWF	PASTU	OAK	OTHERS
ASPEN	324	276	11	22	0	0	15
PPINE	664	0	462	73	8	19	2
DWF	996	44	52	899	0	0	1
PASTU	120	0	19	3	87	11	0
OAK	18	0	4	0	2	11	1

Ave. Performance By Class = 78.8

(a) Results at Iteration 5

	SAMP.	ASPEN	PPINE	DWF	PASTU	OAK	OTHERS
ASPEN	324	292	6	18	0	0	0
PPINE	664	0	567	77	8	12	0
DWF	996	41	45	909	0	0	1
PASTU	120	0	19	3	87	11	0
OAK	18	0	4	0	2	11	1

Ave. Performance By Class = 80.1

(b) Results at Iteration 10

	SAMP.	ASPEN	PPINE	DWF	PASTU	OAK	OTHERS
ASPEN	324	296	6	18	0	0	4
PPINE	664	0	568	78	8	10	0
DWF	996	35	42	918	0	0	1
PASTU	120	0	18	3	87	12	0
OAK	18	0	4	0	2	11	1

Ave. Performance By Class = 80.5

(c) Results at Iteration 20

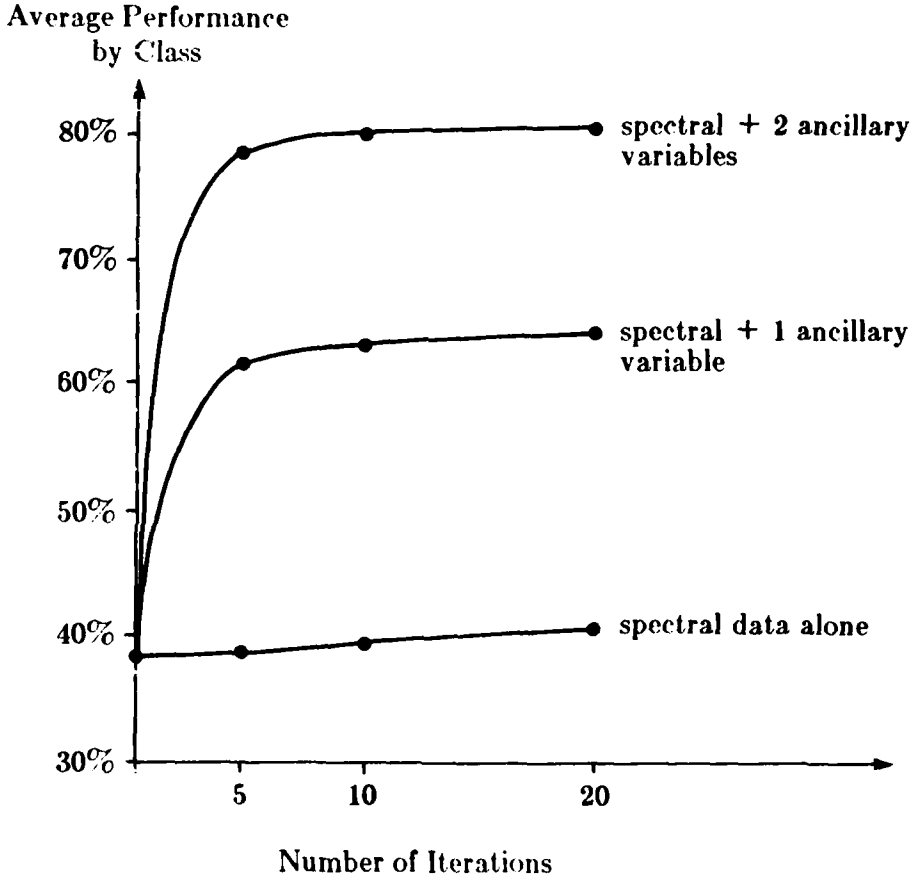


Fig. 2.4.1 Results from Several Different Algorithms

accuracy in classification can be achieved. The probabilistic relaxation methods provides an effective approach for integrating information from diverse sources of image data.

For the present study, there is no specific way to determine the value of weighting constant β in Equation (2.1.6). In any specific image to be classified, the significance of the ancillary data will depend on its relevance and accuracy. Consequently, the optimum degree of supervision must be estimated using training data just as training data are used in establishing classifier parameters.

The algorithm is terminated after the fixed points have been reached, that is, when the likelihoods assigned to each class at every pixel do not change when the algorithm moves from current iteration to next iteration. For the current study, after 20 iterations the algorithm had reached the fixed points. The final labeling represents a balance in consistency between spectral information in the initial likelihoods $P_i^{(0)}(\lambda)$, spatial context data incorporated in $P_{ij}(\lambda \mid \lambda')$, and ancillary information embedded in the $\psi_i(\lambda)$.

2.5 Summary and Conclusions

The relaxation operator has been adopted as a mechanism for incorporating contextual information into an existing classification result. Based on the formula derived in Equation (2.1.4), the supervised relaxation operator carriers on further and incorporates information from multiple ancillary data sources into the results of an existing classification, i.e., to integrate information from the various sources, reaching a balance in consistency between spectral, spatial, and ancillary data sources of information.

In Section 2.1, the supervised relaxation algorithm was derived from the standard relaxation formula to incorporate ancillary information by adjusting the neighborhood contribution to contain the influences from both local context and ancillary information. This means that the moving direction of the initial likelihoods is influenced now by both local context and ancillary information as shown in Fig. 2.3.4. The proof that Algorithm 2.3.2 stops at consistent labelings can be used for the supervised relaxation algorithm by generalizing the support (or neighborhood function) not only from local contextual information but also from any other information, which is useful to improve the classification accuracy, such as ancillary information described in Section 2.4. Thus, the final labeling results from convergence of evidence, reaching consistent labelings, i.e., integrates information from the various sources, achieving a ground cover classification which is both accurate and consistent in the face of inconsistencies which may exist among the data components.

Section 2.4 showed experimental results of the supervised relaxation algorithm. With the contextual information incorporated in the relaxation algorithm, the performance was slightly better than that obtained from the maximum likelihood classifier. By incorporating one ancillary information source, channel 3 data, using supervised relaxation algorithm, the performance was much better than previously obtained. For the area classified, there were data available describing the elevation preferences of the various tree species, along with a digitized elevation map. By incorporating both elevation data and channel 3 data in the supervised relaxation algorithm, significantly better

performance was obtained. These results demonstrate that the supervised relaxation algorithm is a useful technique for incorporating both contextual information and multiple ancillary information sources into an existing classification.

It will often be the case that one has only a classification map to work with rather than sets of likelihoods generated by a classification algorithm. Under this circumstance, arbitrary "probability" values consistent with the classification can be assigned to classes at every object as the initial likelihoods. Of course, the label for a particular object indicated by the classifier as most likely must be assigned the largest probability value. But for the present study, the initial likelihoods, $P_i^{(0)}(\lambda)$, $\lambda \in \Lambda$, were available in the penultimate step of the maximum likelihood classification using spectral data only. Therefore, classification results with higher accuracy may be expected since pixels or objects with only marginal likelihoods from the maximum likelihood classifier may have their classes or labels changed early in the process rather than having them fixed erroneously as a result of the initial likelihoods assigned. As seen from the classification results, the distributions of various classes tend to be more homogeneous than that before relaxation. This is the characteristic of the relaxation operator showing that the spatial context information has been used.

At the moment, there is no specific way to determine the value of weighting constant β, in Equation (2.1.6), through which the supervised relaxation algorithm allows the relative influence of spectral and ancillary data sources to be varied. In any particular image to be classified, the significance of the ancillary data will depend on both its relevance and accuracy. Consequently, the optimum degree of supervision, i.e., weighting to be given to

the influence of the ancillary data via the parameter β, must be estimated using training data just as training data are used in establishing classifier parameters. Obviously, this involves additional analysis cost.

The contributions from the four nearest neighbors have been used in the supervised relaxation algorithm for the present study to incorporate the contextual information into an existing classification. For the simulation presented in Section 2.4, equal weighting constants, d_{ij}, have been assigned to the four nearest neighbors; i.e., these four neighboring pixels have equal degree of influence in the neighborhood contribution. If the weighting constant, d_{ij}, can be dynamically adjusted to allow different neighbors to have different degrees of influence on the current pixel classification, the classification accuracy expected may be better than that with fixed weighting constants. Clearly, there is a tradeoff between the better classification accuracy and the cost involved in dynamically adjusting weighting constants.

In the integration of multiple ancillary data sources, the elementwise product of sets of likelihoods derived from distinctive ancillary data sources is used. This means that each distinctive ancillary data source is given equal degree of confidence. In a more complicated case such as one ancillary data source being more reliable than the other, it may be desirable to assign two different weighting constants to the different data sources.

To use the supervised relaxation as a post classification technique, the set of context conditional likelihoods $P_{ij}(\lambda \mid \lambda')$ must be determined. The required context conditional likelihoods, $P_{ij}(\lambda \mid \lambda')$, can be estimated from the results of the maximum likelihood classifier if no other spatial model known to be correct for the image under consideration is available. Obviously the results will suffer some inaccuracy because the conditional likelihoods are estimated from the

results which are not perfectly correct. For the simulation results presented here, the contextual conditional likelihoods were estimated from another source of data known to be correct. When the number of possible classes involved is large, the number of $P_{ij}(\lambda | \lambda')$ values is also large. Based on the work in [20], it is suggested that highly accurate compatibilities are not required. Therefore, reasonable values can be estimated from the results of the maximum likelihood classifier or from some foreknowledge of the spatial characteristics of the image data if the initial likelihoods in the penultimate step of the classification algorithm are not available.

CHAPTER 3 - PARALLEL PROCESSING

This chapter will describe how an optimal system configuration can be determined based on the performance criteria and parallel architecture described in this chapter. An SIMD (Single-Instruction-stream, Multiple-Data-stream) machine, as shown in Fig. 3.1, consists of a control unit, N processors, N memory modules, and an interconnection network. A processor and a memory module forms a processing element (PE). The control unit broadcasts instructions to all active PEs and all active PEs execute the same instruction, *in lock-step*, on the data in their own memories. The interconnection network provides a communication facility for the PEs. An MIMD (Multiple-Instruction-stream, Multiple-Data-stream) machine, as shown in Fig. 3.2., consists of a coordinator, N PEs, and an interconnection network. Every PE fetches instructions from its own memory and executes them on the data in its own memory. Every PE executes the instructions independently and asynchronously. The interconnection network provides a communication medium for the PEs. The coordinator orchestrates the activities of the PEs.

Section 3.1 will present the application of S-Nets [14,15] to describing an SIMD and pipeline implementation of maximum likelihood classification, and several performance measures to evaluate the inherent parallelism in this algorithm will be discussed in this section. In Section 3.2, for the algorithm in block C shown in Fig. 3.2.1 the algorithm execution times based on the

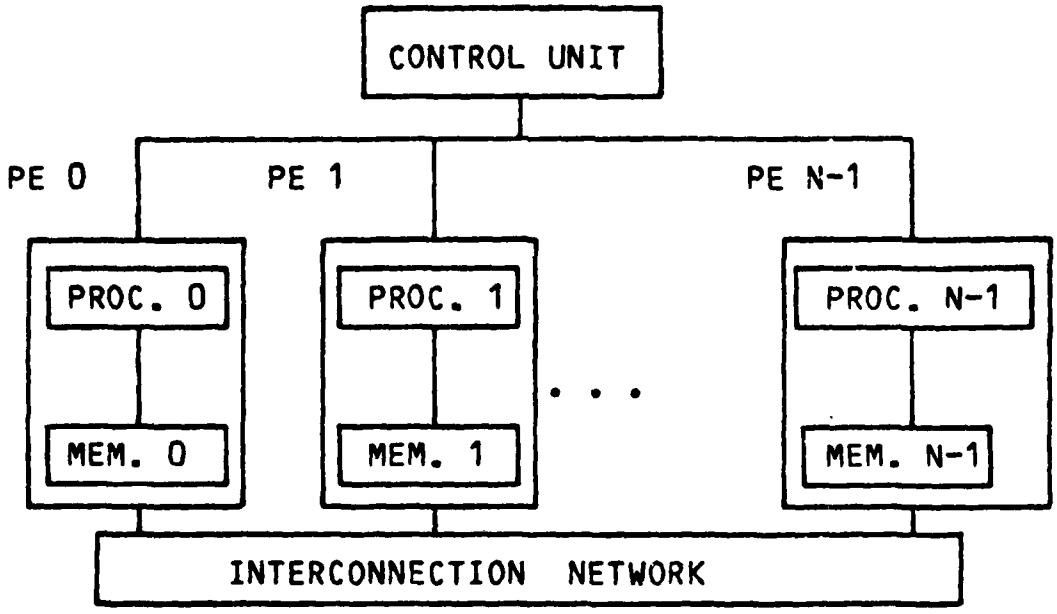


Fig. 3.1. SIMD [26]

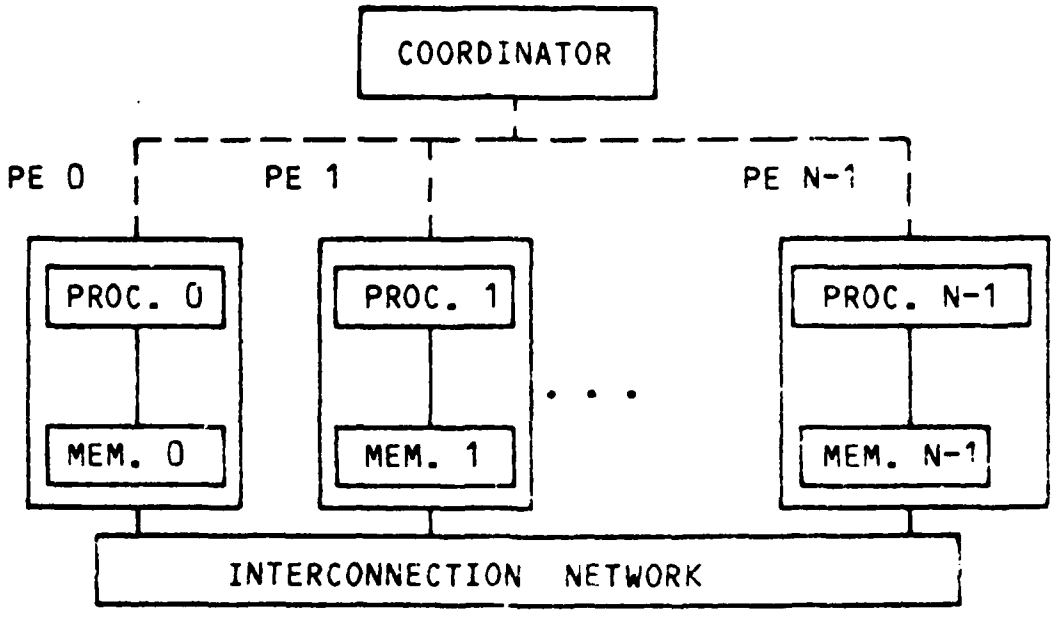


Fig. 3.2. MIMD [26]

counting of explicit and implicit operations in both SIMD and MIMD mode of parallelism will be compared and discussed. In Section 3.3, several performance measures different from those in Section 3.1 for SIMD will be discussed. Most important is the determination of the optimal number of processing elements for a given image size by considering the tradeoff between the cost of execution time and processing elements. Section 3.4 will describe how two multistage networks [16,17,18,19], cube network and ADM network, can be configured to support simultaneously both MIMD and SIMD modes of parallelism.

3.1 Modeling Image Classification by Means of S-Nets

Synchronous Nets [14,15], hereafter S-Nets, are an extension of Petri nets and were developed especially for the description of SIMD processes. This section presents the application of S-Nets to describing maximum likelihood classification, which is commonly used for classifying remote sensing image data. This application is fairly typical of image processing operations which are *not* window-type operations, i.e., they depend on the data at a single pixel rather than a neighborhood. In general, the higher the dimensionality of the remote sensing (multispectral) data and the more classes represented in the image, the greater the potential benefits to be derived from SIMD implementation of the process. This section begins with an introductory overview of S-Nets.

3.1.1 S-Net Structure: Overview [14]

S-Nets are defined in terms of sets. The elements of a set are designated within { }. The notion of tuples is denoted by < >; a tuple consists of ordered components. An S-Net graph is a quadruple (T,S,U,A), with an initial marking K_0 and a set of transition descriptors D, where:

T = A finite set of transitions $\{t_1,t_2, \ldots, t_{|T|}\}$.

S = A finite set of scalar places $\{s_1,s_2, \ldots, s_{|S|}\}$.

U = A finite set of vector mask places

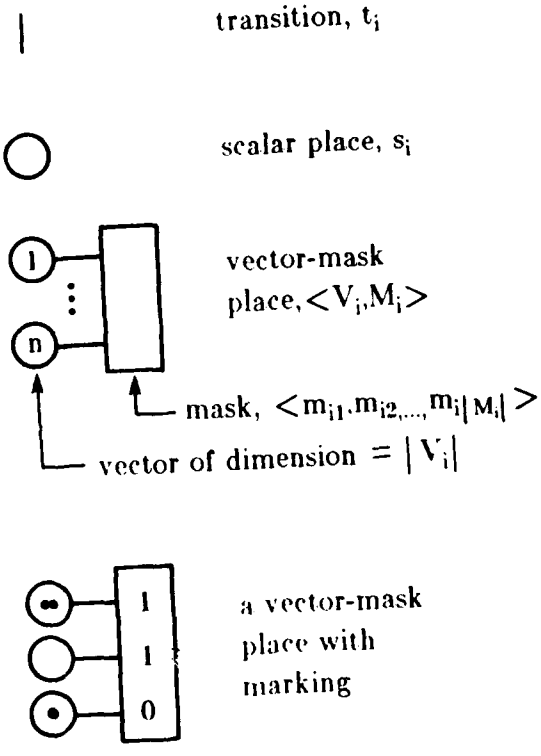
$\{<V_1,M_1>,<V_2,M_2>,\ldots,<V_{|U|},M_{|U|}>\}$.

A = A finite set of directed arcs $\{a_1,a_2,\ldots,a_{|A|}\}$.

Additional symbols utilized in constructing S-Net graphs are shown in Table 3.1.1.

A marking associates a non-negative integer with each scalar place - K(s) for each $s \epsilon S$; and two vectors of non-negative integers with each vector-mask place, one of these vectors being a boolean vector $K(V_i)$, for each V_i, and $K(M_i)$ for each M_i. An initial marking K_0 is defined as the first marking of the S-Net. The set of possible mask markings for any M_i is $W(M_i)$. The marking <1,1,0,0,0> of M_i shall be denoted as $<1^2,0^3>$. In S-Net graphs, markings are indicated by the presence of tokens. Dots in any place represent tokens. The symbol 1 shows the presence of a token in a mask. An assignment of tokens to a vector place V_i may leave some of the component places marked with tokens and others empty. The dynamic behavior of S-Nets is described as follows.

Table 3.1.1. Symbols Used in S-Net Graphs



A scalar place is ***holding*** if it has at least one token in it. A vector-mask place $<V_i,M_i>$ is holding if at least one $K(m_{ij}) = 1$, $j=1,2,\ldots,|M_i|$, and the corresponding $v_{ij} \epsilon V_i$ has a non-zero marking. A transition t is *enabled* if all scalar places of transition t are holding and all vector-mask places of transition t are holding. When a transition t is enabled, its firing function is defined at a given marking K_n of the S-Net, and the firing yields K_{n+1}, a new marking. A transition type specifies the firing capabilities of the transition - either simple or mask firing - designated SFT and MFT respectively. A transition descriptor D[t] for a transition t with vector-mask output places $<V_i,M_i>, <V_j,M_j>, \ldots, <V_r,M_r>$ is specified as D[t] = [type; $K(M_i)\epsilon W(M_i), K(M_j)\epsilon W(M_j), \ldots, K(M_r)\epsilon W(M_r)$].

A ***simple firing*** associated with an enabled transition t is such that:

(a) For every scalar input place s, $K_{n+1}(s) = K_n(s)-1$.

(b) For every scalar output place s, $K_{n+1}(s) = K_n(s)+1$.

(c) For every vector-mask input place $<V_i,M_i>$, then for $v_{ij}\epsilon V_i$, $j=1,2,\ldots,|V_i|$, $K_{n+1}(v_{ij}) = K_n(v_{ij})-1$ for those j for which $m_{ij}\epsilon M_i$ has a non-zero marking; and for $m_{ij}\epsilon M_i$, $K_{n+1}(m_{ij}) = K_n(m_{ij})$ for all j.

(d) For every vector-mask output place $<V_i,M_i>$, then for $v_{ij}\epsilon V_i$, $j=1,2,\ldots,|V_i|$, $K_{n+1}(v_{ij}) = K_n(v_{ij})+1$ for those j for which $m_{ij}\epsilon M_i$ has a non-zero marking; and for $m_{ij}\epsilon M_i$, $K_{n+1}(m_{ij}) = K_n(m_{ij})$ for all j.

As seen from the firing rules, SFTs do not alter their input or output masks.

A mask firing is associated with an enabled transition t that has at least one $<V_i,M_i>$ output place, and is such that:

(a) For every scalar input place s, $K_{n+1}(s) = K_n(s)-1$.

(b) For every scalar output place s, $K_{n+1}(s)=K_n(s)+1$.

(c) For every vector-mask input place $<V_i,M_i>$, then for $v_{ij}\epsilon V_i$, $j=1,2,...,|V_i|$, $K_{n+1}(v_{ij}) = K_n(v_{ij})-1$ for those j for which $m_{ij}\epsilon M_i$ has a non-zero marking; and for $m_{ij}\epsilon M_i$, $K_{n+1}(m_{ij}) = K_n(m_{ij})$ for all j.

(d) For every vector-mask output place $<V_i,M_i>$, then for $v_{ij}\epsilon V_i$, $j=1,2,...,|V_i|$, $K_{n+1}(v_{ij}) = K_n(v_{ij})+1$ for those j for which $m_{ij}\epsilon M_i$ has a non-zero marking; and for M_i, $K_{n+1}(M_i)\epsilon W(M_i)$, where $W(M_i)$ is specified by the transition descriptor D[t], and $K_{n+1}(M_i)$ is non-deterministically chosen.

By the firing definitions, firings remove tokens from some places and add tokens to other places. However, the number of tokens subtracted by a transition firing does not necessarily equal the number that it adds.

SFTs on firing do not change the $K(M_i)$ of their $<V_i,M_i>$ input and output places; the markings for any output masks of the transition are specified at K_o. This transition descriptor is noted simply as D[t] = [SFT; __].

For MFTs, the $|W(M_i)| \geq 1$ for all output masks. These markings are accomplished by the transition firing and after initial marking, and the set of markings must be listed in the transition descriptor, i.e., D[t] = [MFT; $K(M_i) \epsilon \{< >, < >,...,< >\}$]. In cases where the set of markings range over the $|M_i|$-fold Cartesian product of the boolean set, then:

$$D[t] = [MFT; K(M_i) \epsilon B^{|M_i|}], \text{ where B is the set of boolean numbers } \{0,1\}.$$

An S-Net is *safe* iff $K(s) \leq 1$ for all $s \epsilon S$; $K(v_{ij}) \leq 1$ for all $v_{ij} \epsilon V_i$.

An S-Net is *consistent* iff for all $<V_i,M_i>$ places of the net, no $K(v_{ij}) > 0$ when $K(m_{ij}) = 0$.

3.1.2. Measures of Parallelism [14]

This section summarizes the quantitative measures of concurrency in S-Nets, both with and without the context of time. These are called measures of the degree of parallelism.

The Degree of Vector Parallelism $\hat{g}_n$ of a transition t_n is defined as the sum of the number of tokens fired by the transition into V_i of every $<V_i,M_i>$ output place of the transition.

The Degree of Parallelism g_n of a transition t_n is defined as the sum of the number of the tokens fired by the transition into its scalar output places and into V_i of every $<V_i,M_i>$ vector-mask output place.

The Average Degree of Vector Parallelism $\bar{\hat{g}}$ over some sequence of transitions $t_n = t_j,t_{j+1}, \ldots, t_k$ is defined as

$$\bar{\hat{g}} = \frac{\sum_{r=j}^{k} \hat{g}_r}{|t_n|}$$

where $|t_n|$ is the total number of transitions in sequence t_n.

The Average Degree of Parallelism $\bar{g}$ over some sequence of transitions $t_j,t_{j+1}, \ldots, t_k$ is defined as

$$\bar{g} = \frac{\sum_{r=j}^{k} g_r}{|t_n|}$$

The Average Vector Parallelism $\bar{\hat{h}}$ achieved over some sequence of non-primitive transitions $t_n = t_j, t_{j+1}, \ldots, t_k$ is defined as

$$\bar{\hat{h}} = \frac{\sum_{r=j}^{k} \hat{g}_r * e_r}{e_n}$$

where e_r is the time units the transition t_r takes to complete its operation, and

$$e_n = \sum_{r=j}^{k} e_r.$$

The Average Parallelism $\bar{h}$ achieved over some sequence of non-primitive transitions $t_n = t_j, t_{j+1}, \ldots, t_k$ is defined as

$$\bar{h} = \frac{\sum_{r=j}^{k} g_r * e_r}{e_n}$$

3.1.3 Stone's Vector Sum Example [14]

The problem is to compute

$$Y_n = \sum_{i=0}^{n} A_i, \qquad n = 0,1,\ldots,7$$

so that
$$\begin{aligned} Y_0 &= A_0 \\ Y_1 &= A_0 + A_1 \\ &\cdot \quad \cdot \quad \cdot \\ &\cdot \quad \cdot \quad \cdot \\ &\cdot \quad \cdot \quad \cdot \\ Y_7 &= A_0 + A_1 + \cdots + A_7 \end{aligned}$$

where $A_0, A_1, \ldots, A_7$ are scalars. Assume there is a SIMD machine of 8 PEs (N=8). A high level language expression of the computation, in simplified

form, is as follows.

1. Initialize Y[I] to A[I], $0 \leq I \leq N-1$
2. For k = 1 step 1 until $\log_2 N$, do
3. Begin: $H[I] = Y[I-2^{k-1}]$, (mask)
4. $Y[I] = Y[I] + H[I]$, (mask)
5. Compute new mask
6. End

Figure 3.1.1, representing the S-Net model of this algorithm, assumes an initial marking:

$K_0(S_1) = 1$; $K_0(S_2) = K_0(S_3) = K_0(S_4) = K_0(S_5) = (0)$;

$K_0(V_1) = K_0(V_2) = K_0(V_3) = <0^8>$;

$K_0(M_1) = <1^8>$.

Descriptors are:

$D[t_1] = D[t_4] = D[t_5] = D[t_7] = [SFT:\_]$;

$D[t_2] = [MFT;K(M_2) = <0,1^7>]$

$D[t_3] = [MFT;K(M_3) \epsilon \{<0,1^7>,<0^2,1^6>,<0^4,1^4>\}]$

$D[t_6] = [MFT;K(M_2) \epsilon \{<0^2,1^6>,<0^4,1^4>\}]$

Statement 1, modeled by transition t_1, indicates that all PEs simultaneously carry out the assignments:

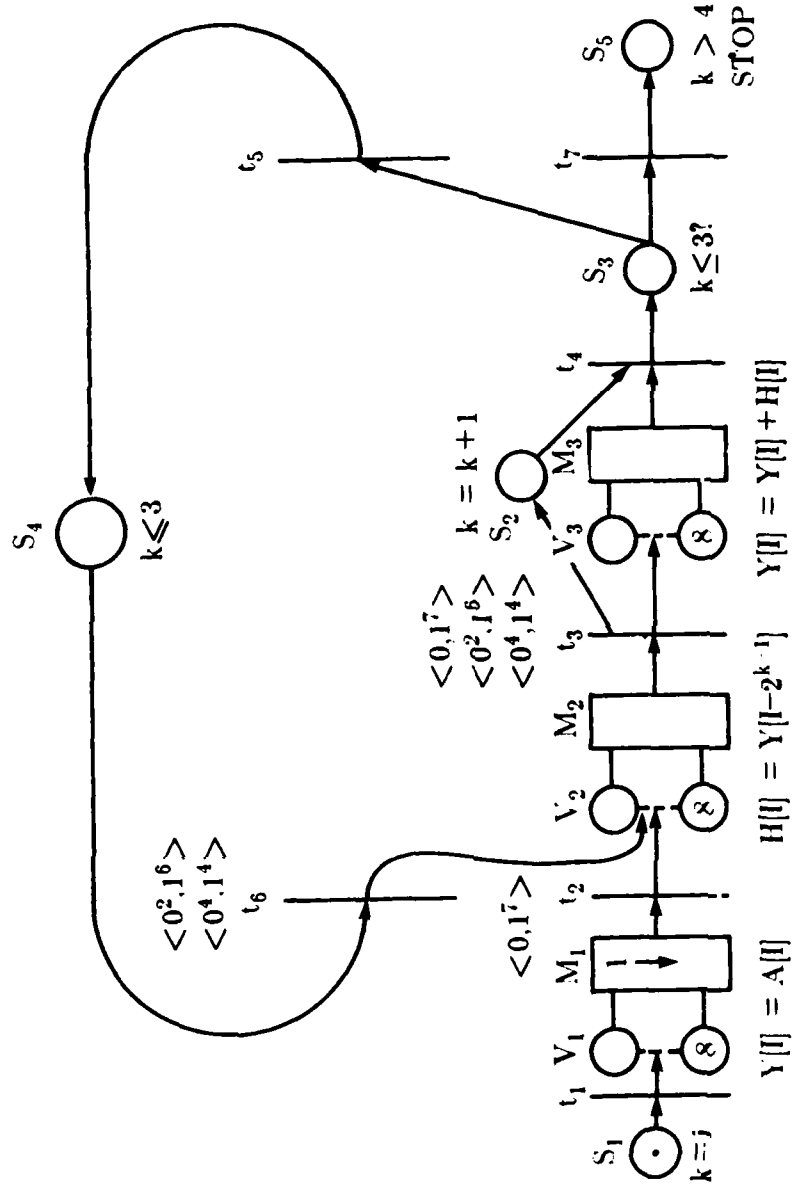


Fig. 3.1.1 S-Net of Vector Sum

$$Y(0) = A(0),\ Y(1) = A(1),\ \ldots,\ Y(7) = A(7).$$

Since each PE has both Y(I) and A(I), these operations can take place "in parallel."

Statement 3, modeled by t_2, requires that data are transferred between participating PEs. For the firing of t_2, H(1) holds Y(0), H(2) holds Y(1),..., and H(7) holds Y(6). PE_0 does not participate, since $K(M_2) = <0,1^7>$.

Statement 4, modeled by t_3, has all participating PEs adding simultaneously. If MFT t_3 first fires $K(M_3) = <0,1^7>$, then for the first iteration of the loop, PE 0 does not participate. The first holding of $<V_3,M_3>$ models sums: Y(1) has Y(0) + Y(1); Y(2) has Y(1) + Y(2), etc.

A branch is modeled by the t_5 and t_7 transition which have scalar input place s_3 in common, modeling the result of a test of the iteration counter.

The firing of t_2 models the first iteration (k=1), the first firing of t_6 models the second iteration (k=2), and the second firing of t_6 models the third iteration (k=3). After the third iteration, transition t_7 will fire and the computation halts, as all eight sums reside in the appropriate Y( ) variables.

Despite the availability of 8 PEs, the degree of parallelism achieved over the computational flow is not 8. Over the firing sequence described and listed in Table 3.1.2, $\bar{g}$ and $\tilde{g}$ are calculated as follows:

$$\bar{g} = 51/13 = 3.92;\ \tilde{g} = 42/13 = 3.23.$$

The maximum parallelism achieved at this level of modeling is reduced by the nature of the algorithm and by the scalar processes modeled in the testing of the loop. The use of the $\bar{h}$ and $\tilde{h}$ measures are not considered here. In a realistic analysis, some weighting of the additional time to accomplish mask

Table 3.1.2. The Measures g and $\hat{g}$ for Vector Sum S-Net

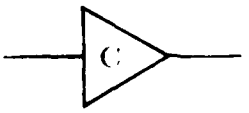
Transition Sequence t_n	g	$\hat{g}$
t_1	8	8
t_2	7	7
t_3	8	7
t_4	1	0
t_5	1	0
t_6	6	6
t_3	7	6
t_4	1	0
t_5	1	0
t_6	4	4
t_3	5	4
t_4	1	0
t_7	1	0
$\lvert t_n \rvert = 13$	51	42

$$\bar{g} = \frac{51}{13} = 3.92 \; ; \; \bar{\hat{g}} = \frac{42}{13} = 3.23$$

firing would be appropriate, as for transitions t_2, t_3, and t_6.

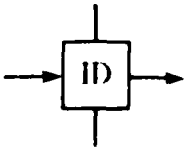
3.1.4. Modeling Complex Applications with S-Nets [14].

S-Nets have been developed to provide a tool to express algorithm implementation on SIMD and MSIMD architectures. The implementations are modeled by using safe and consistent S-Nets. In modeling realistic applications, the first obvious complication in S-Net graphs is that of length, i.e., the length of the transition sequence will not fit within some desired reference frame. Connectors, such as shown here



may be used to show where an arc breaks and reconnects, much in the manner that connectors are used in flow charts. The connector can be inserted between a transition and the following place, or between a place and the following transition.

Macro S-Net transitions (macros) are introduced to support hierarchical modeling and to permit a more abstracted representation of an event which is a component of a computation. A *macro transition* t is defined as a transition which itself is an S-Net graph. It begins with a single vertex t_r and ends with single vertex t_s, where t_r and t_s themselves can be macro S-Net transitions. A label is appended to the transition bar to distinguish macros in the global flow, shown as



where ID is any symbol used by the modeler.

Figure 3.1.2 illustrates hierarchical modeling, defining two macros and showing how the substitution of the greater detail can be modeled in the S-Net.

With this definition, the non-primitive transition t_n (which is a sequence of transition, $t_j, t_{j+1}, \ldots, t_k$) is seen as a type of macro since it represents a transition sequence beginning with t_j and terminating with t_k.

3.1.5 Modeling Maximum Likelihood Classification with S-Nets

The purpose of this section is to use S-Nets to model an SIMD implementation of maximum likelihood classification [21]. Assume that there are N=M PEs available where the image size is M-by-M. The PEs can be arranged in a row of M elements or in a $\sqrt{M}$-by-$\sqrt{M}$ array. Each PE will be assigned *either one image column* or a $\sqrt{M}$-by-$\sqrt{M}$ subimage that will be stored in its memory.

The SIMD implementation of maximum likelihood classification has been described in [22]. For each of the m possible classes into which a pixel may be classified, a discriminant function is computed. Each discriminant value depends on the pixel vector, the corresponding class mean vector and covariance matrix, and a constant related to the prior probability of occurrence of the class. The pixel is assigned to the class yielding the largest discriminant value.

The essential calculations are as follows. Let

$X = \left[x_1, x_2, \ldots, x_n\right]^T$, the n-dimensional pixel vector

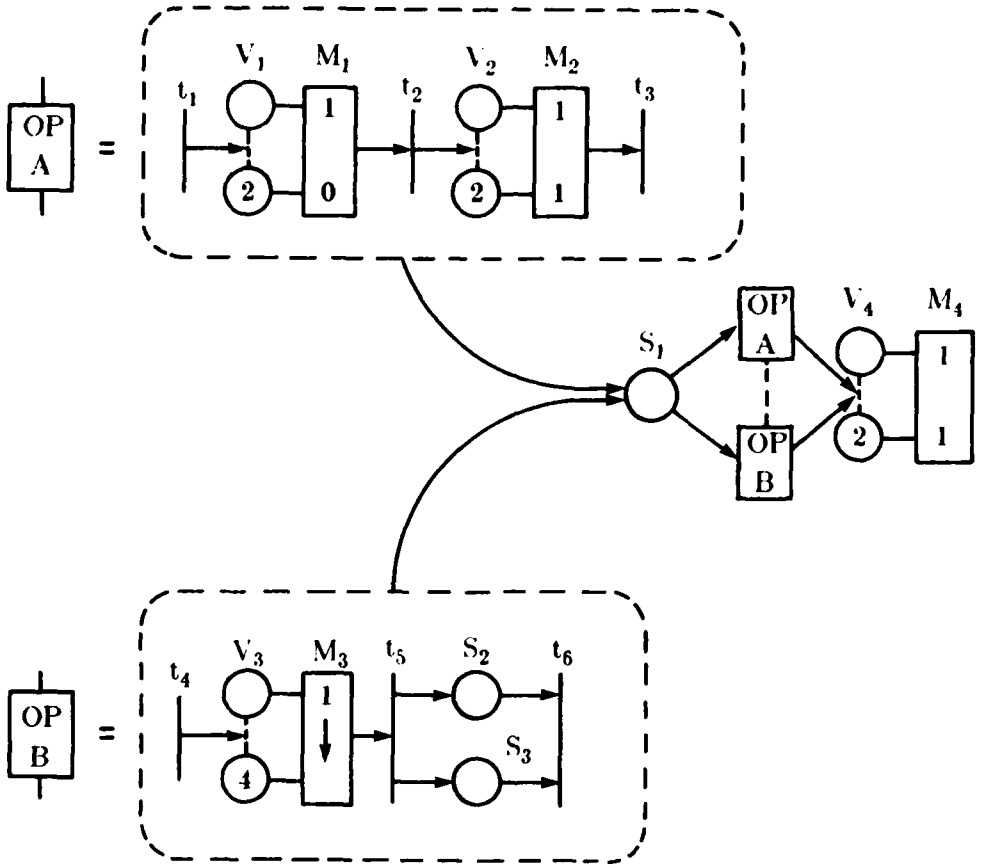


Fig. 3.1.2 Hierarchical Modeling with S-Net Macro Transitions [14]

$U_i = \left[u_1^i, u_2^i, \ldots, u_n^i\right]^T$, the n-dimensional mean vector for class i

$$\sum_i^{-1} = \begin{bmatrix} e_{11}^i & e_{12}^i & \ldots & e_{1n}^i \\ e_{21}^i & & \vdots & \\ \vdots & & & \\ e_{n1}^i & \ldots & & e_{nn}^i \end{bmatrix}$$

the n-by-n inverse of the covariance matrix for class i

c_i = constant pertaining to class i

In addition, A(i) will be the discriminant value for class i. Figure 3.1.3 shows how these values are calculated and a pixel is classified. The following conditions are assumed:

1. All pixels are to be stored in the PE memories. Each PE is responsible for M pixels.
2. Mean vectors (n-dimensional) and inverse covariance matrices (n-by-n) of m classes are stored in all PE memories. The m class constants c_i are also stored in all PE memories.
3. Concurrency between the scalar host and the array resources can be supported by the hardware. The scalar host can be regarded as the control unit and the array resources as the processors.

The computation proceeds as follows:

1. Initialize the values of A(i) to constant c_i, for $1 \leq i \leq m$.
2. Compute the values of the discriminant functions for all m classes.

```
/* Phase A - initialize the discriminant values A(i) */
   for i ← 1 to m
     do A(i) ← c_i
   end

/* Phase B - calculate the discriminant values for each class */
   for ℓ ← 1 to m
     for i ← 1 to n
       do y_i ← x_i − u_i^ℓ
          z_i ← 0
     end
     for j ← 1 to n
       for k ← 1 to n
         do z_j ← z_j + c_jk^ℓ * y_k
       end
       A(ℓ) ← A(ℓ) − y_j * z_j
     end
   end

/* Phase C - find maximum discriminant value and class */
   j ← 1
   for i ← 2 to m
     do if A(i) > A(j) then j ← i
   end
```

Fig. 3.1.3. Maximum Likelihood Classification of a Pixel

3. Choose the maximum value of the discriminant functions.
4. If there are more data in PEs, then go to step 1.
5. End

The S-Net modeling this computation is shown in Figure 3.1.4. There are three macros, INIT A, DIS B and MAX C, defined in Figures 3.1.5, 3.1.6, and 3.1.7, respectively. A brief description of the S-Net is as follows:

- The dimensionality of the V_i components, $|V_i| = M$, represents the number of PEs available in this SIMD implementation.
- The initial marking of the S-Net is

$K_0(S_1) = (1)$;
$K_0(S_2) = K_0(S_3) = \ldots\ldots = K_0(S_{28}) = (0)$;
$K_0(V_1) = K_0(V_2) = \ldots\ldots = K_0(V_7) = <0^M>$;
$K_0(M_1) = K_0(M_2) = \ldots\ldots = K_0(M_6) = <1^M>$;

- Descriptors are:

$D[t_1] = D[t_2] = \ldots\ldots = D[t_{26}] = [SFT;\_]$;
$D[t_{27}] = [MFT;K(M_7) \in B^M]$;
$D[t_{28}] = D[t_{29}] = \ldots\ldots = D[t_{33}] = [SFT;\_]$.

- S_1 has a token in Figure 3.1.4 initially, so the INIT A macro transition is enabled and begins execution when its first transition t_1 fires.

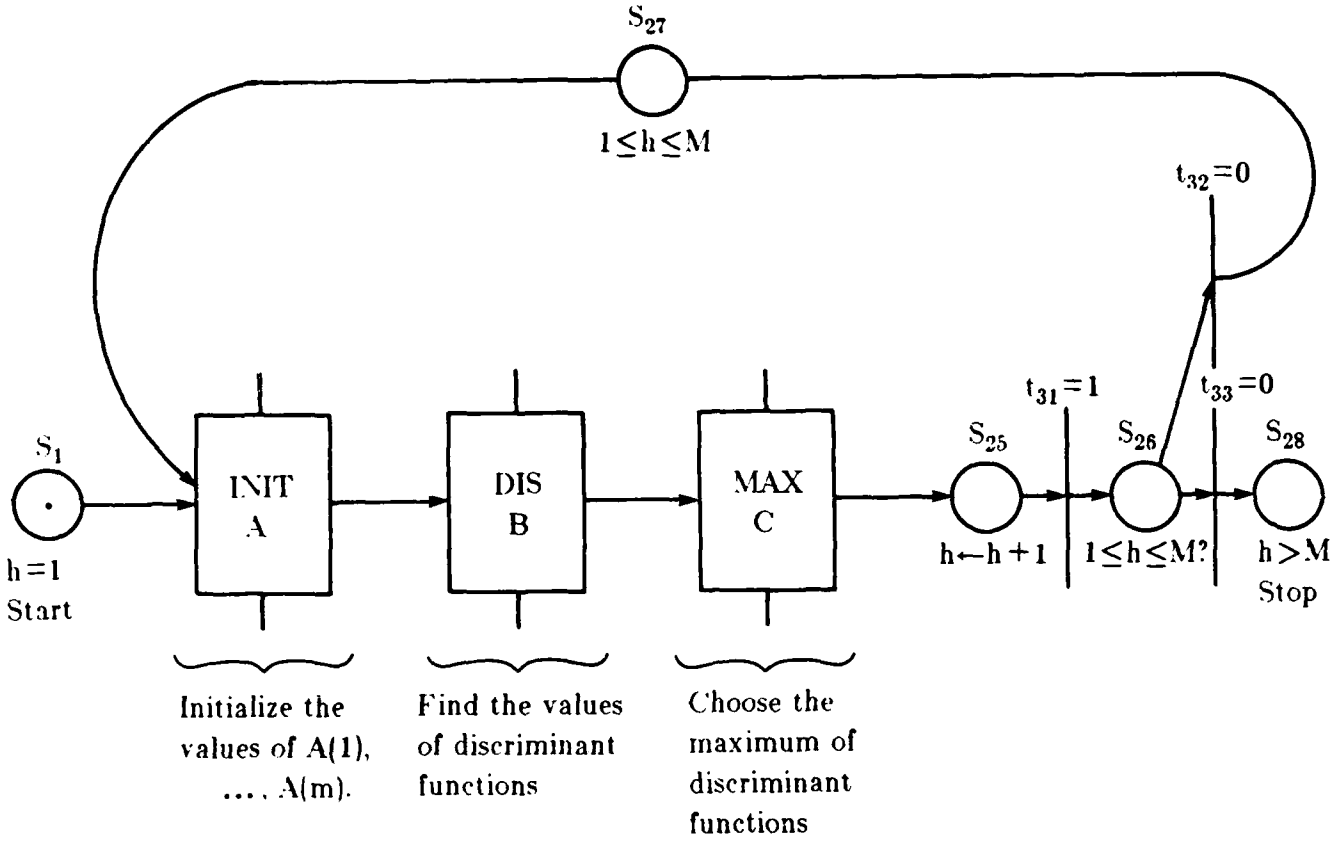


Fig. 3.1.4 S-Net Model of Maximum Likelihood Classification

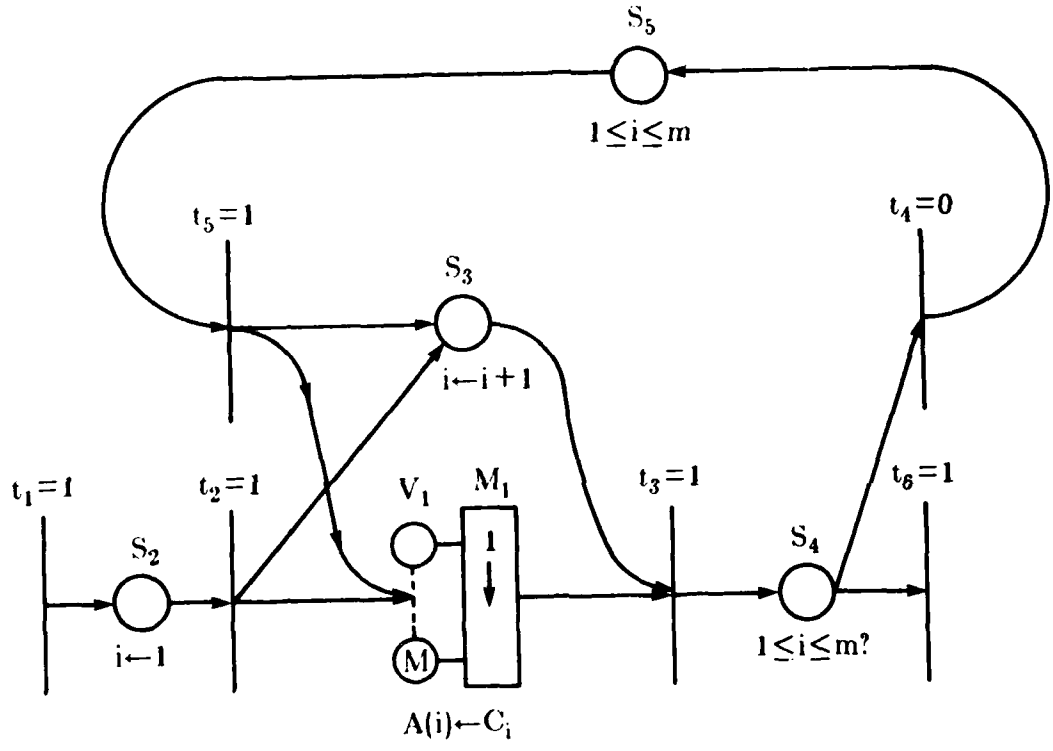


Fig. 3.1.5. Macro INIT A - Initialize the Values of A(1),...,A(m)

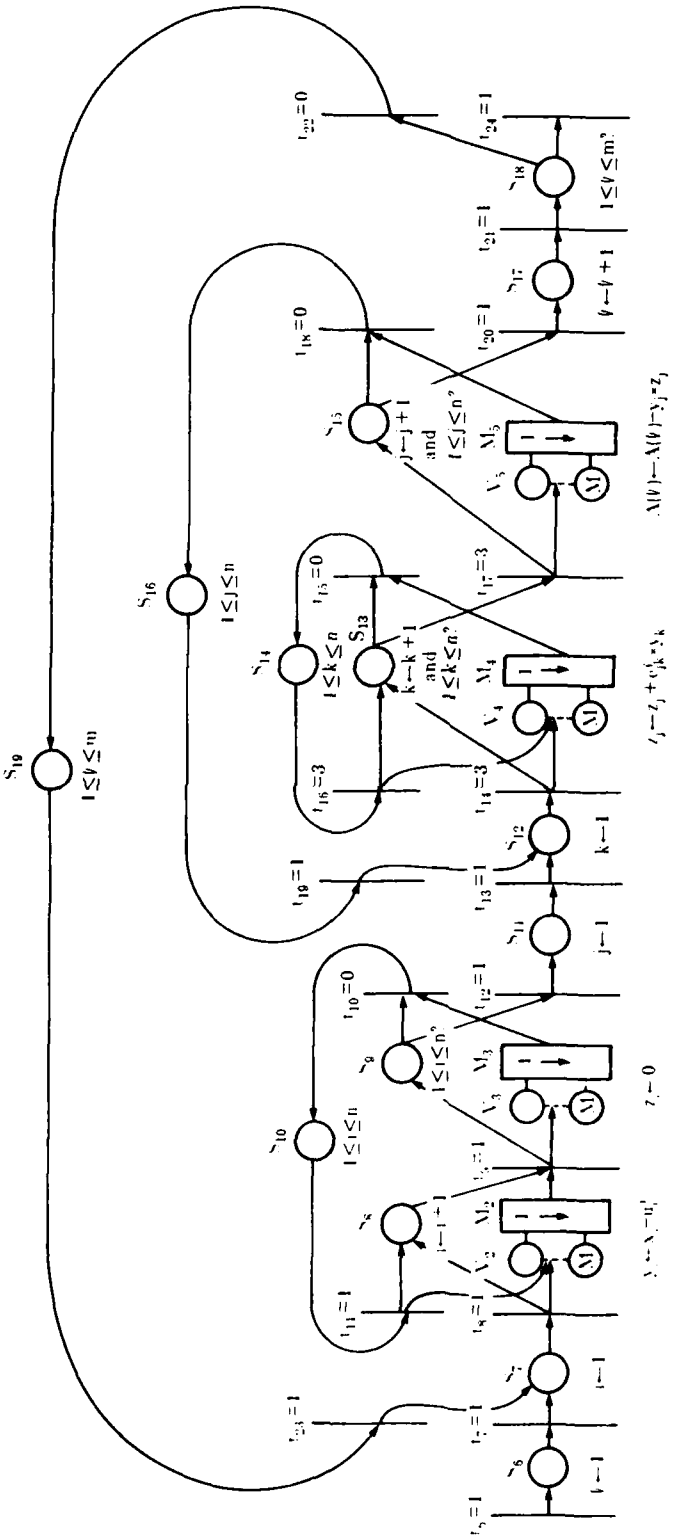


Fig. 3.1.6 Macro DIS B - Find the Values of Discriminant Functions

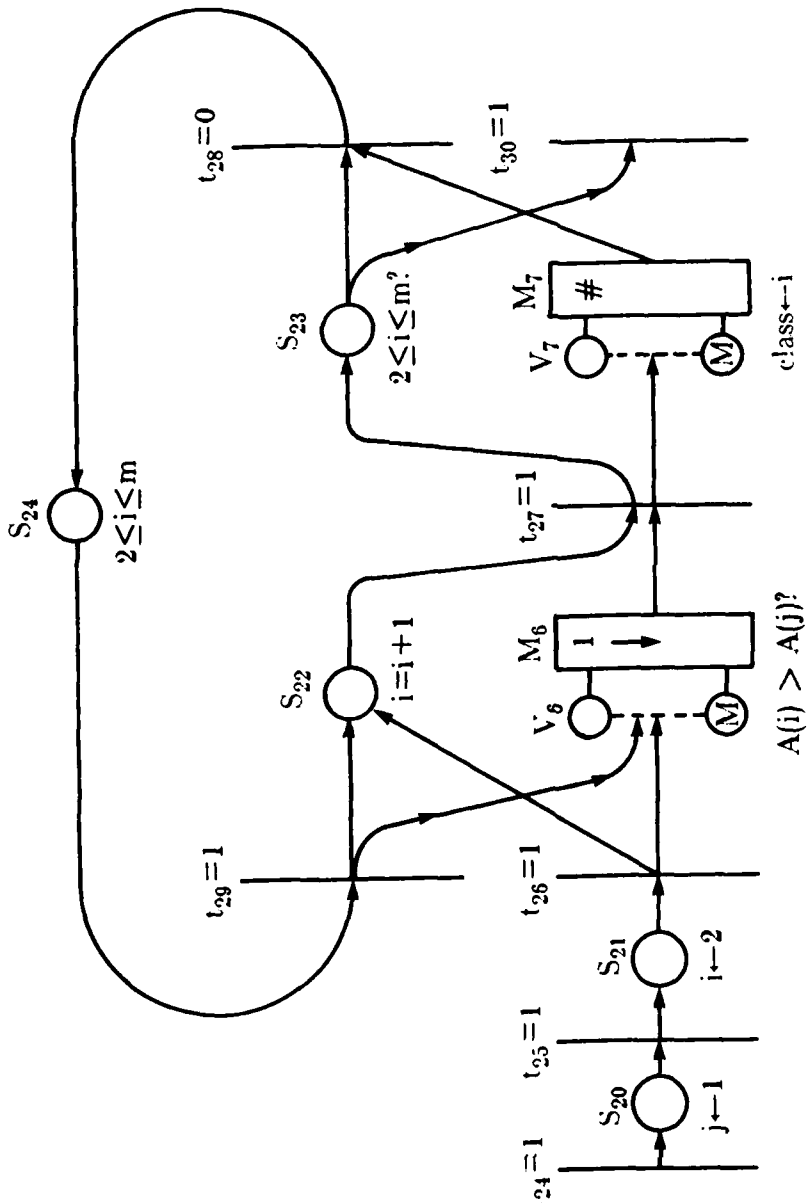


Fig. 3.1.7 Macro MAX C - Choose the Maximum of Discriminant Functions

- The transition t_2 models the initialization of A(i) to a constant, c_i, $1 \leq i \leq m$, with all PEs being active because $|V_1| = M$ and $K(M_1) = <1^M>$, and the execution of the control-flow or scalar instructions in the control unit.
- The transition t_3 models the event: "check if i is greater than m or not."
- The transition t_{26} models a test performed with all PEs. From this test, a mask marking is implicitly formed.
- $K(M_7)$ is marked on the graph as # which means it is data dependent.
- To make timing analyses more realistic, different time units are assigned to the transitions depending on the operations they represent. The notation at each transition is "t_i = time units."

Because of the data dependent condition, an execution of this S-Net over the sequence of transitions in the MAX C macro will not reveal the number of tokens fired. The degree of parallelism is also data dependent. (This difficulty of data dependence is not exclusive to S-Nets; it must be dealt with in any modeling analysis.) In examining the model for synchronization between the stages of S-Nets and measuring the degree of parallelism, we assume all PEs are active.

The results of an execution of the S-Net from t_1 to t_{33} are shown in Table 3.1.3. From this table, the quantitative measures of parallelism are calculated as follows:

Table 3.1.3. Timing Analysis of SIMD S-Net

transition t_i	no. of firings n_i	time the transition t_i takes, e_i	$\hat{g}_i$	g_i	n_i*e_i	$\hat{g}_i*n_i*e_i$	$g_i*n_i*e_i$
t_1	M	1	0	1	M	0	M
t_2	M	1	M	1+M	M	M^2	(1+M)M
t_3	mM	1	0	1	mM	0	mM
t_4	(m−1)M	0	0	1	0	0	0
t_5	(m−1)M	1	M	1+M	(m−1)M	$(m-1)M^2$	(1+M)(m−1)M
t_6	M	1	0	1	M	0	M
t_7	M	1	0	1	M	0	M
t_8	mM	1	M	1+M	mM	mM^2	(1+M)mM
t_9	nmM	1	M	1+M	nmM	nmM^2	(1+M)nmM
t_{10}	(n−1)mM	0	0	1	0	0	0
t_{11}	(n−1)mM	1	M	1+M	(n−1)mM	$(n-1)mM^2$	(1+M)(n−1)mM
t_{12}	mM	1	0	1	mM	0	mM
t_{13}	mM	1	0	1	mM	0	mM
t_{14}	nmM	3	M	1+M	3nmM	$3nmM^2$	(1+M)3nmM
t_{15}	(n−1)nmM	0	0	1	0	0	0
t_{16}	(n−1)nmM	3	M	1+M	3(n−1)nmM	$3(n-1)nmM^2$	(1+M)3(n−1)nmM
t_{17}	nmM	3	M	1+M	3nmM	$3nmM^2$	(1+M)3nmM
t_{18}	(n−1)mM	0	0	1	0	0	0
t_{19}	(n−1)mM	1	0	1	(n−1)mM	0	(n−1)mM
t_{20}	mM	1	0	1	mM	0	mM
t_{21}	mM	1	0	1	mM	0	mM
t_{22}	(m−1)M	0	0	1	0	0	0
t_{23}	(m−1)M	1	0	1	(m−1)M	0	(m−1)M
t_{24}	M	1	0	1	M	0	M
t_{25}	M	1	0	1	M	0	M
t_{26}	M	1	M	1+M	M	M^2	(1+M)M
t_{27}	(m−1)M	1	M	1+M	(m−1)M	$(m-1)M^2$	(1+M)(m−1)M
t_{28}	(m−2)M	0	0	1	0	0	0
t_{29}	(m−2)M	1	M	1+M	(m−2)M	$(m-2)M^2$	(1+M)(m−2)M
t_{30}	M	1	0	1	M	0	M
t_{31}	M	1	0	1	M	0	M
t_{32}	M-1	0	0	1	0	0	0
t_{33}	1	0	0	1	0	0	0

$$E_{tot} = \sum_{i=1}^{33} n_i * e_i = [4+8m+6mn+3mn^2]M \quad \text{time units}$$

$$\sum_{i=1}^{33} \hat{g}_i * n_i * e_i = [-2+3m+5mn+3mn^2]M^2$$

$$\sum_{i=1}^{33} g_i * n_i * e_i = [4+8m+6mn+3mn^2-2M+3mM+5mnM+3mn^2M]M$$

There are two types of quantitative measure for concurrency.

i) The average vector parallelism $\bar{h}$ only accounts for the concurrency of processing elements; it does not consider the overlap of array and scalar or control-flow instructions.

$$\bar{h} = \frac{\sum_{i=1}^{33} \hat{g}_i * n_i * e_i}{E_{tot}} = \frac{[-2+3m+5mn+3mn^2]M^2}{[4+8m+6mn+3mn^2]M}$$

Typical values for a remote sensing application are n=4, m=16, M=1024. Plugging these values into the above equation, we get the utilization of the M PEs is

$$\frac{\bar{h}}{M} = \frac{1134}{1284} = 0.883$$

i.e., the utilization of the 1024 PEs is 88.3%.

ii) The average parallelism $\bar{h}$ accounts for both concurrency of processing elements and for the overlap of array and scalar or control-flow instructions.

$$\bar{h} = \frac{\sum_{i=1}^{33} g_i * n_i * e_i}{E_{tot}}$$

$$= \frac{(4+8m+6mn+3mn^2)M}{(4+8m+6mn+3mn^2)M} + \frac{(-2+3m+5mn+3mn^2)M^2}{(4+8m+6mn+3mn^2)M}$$

Using the typical values of m, n, and M in the above equation, the utilization of the M PEs is

$$\frac{\bar{h}}{M} = \frac{1}{1024} + \frac{1134}{1284} = 0.884$$

By this measure, the utilization of the 1024 PEs is 88.4%. Since 1024 processing elements are available, the overlap of the control unit with the array processors contributes very little to the degree of parallelism. If the number of processing elements were much less than what we have here, the overlap of the control unit with the array processors would be more significant.

3.1.6. Modeling a Pipeline Implementation of Maximum Likelihood Classification

In this section, a pipeline implementation of maximum likelihood classification will be modeled with S-Nets. Recall that there are N=M PEs available. The PEs will be interconnected to form a set of parallel pipelines each operating on its own subimage. To enable direct comparisons, the proposed architectures will have the same total number of identical processing elements. The following conditions are assumed:

1. The n-dimensional mean vector and n-by-n inverse covariance matrix of class i have been stored in PE_{i-1}, as have the m class constants.
2. The parallelism being exploited has each stage concurrently performing a different step of the task on a different data item (pixel).
3. Assume $\frac{M}{m}$ is an integer and that there are $\frac{M}{m}$ pipelines. Each pipeline will process mM pixels.
4. In calculating the degree of parallelism, the overhead required for the pipeline to reach full efficiency will be ignored, i.e., steady state operation is assumed.

A high level description of this computation at stage i−1 is as follows:

1. Initialize the value of A(i) to constant c_i.
2. Find the value of the discriminant function for class i.
3. Compare the value of the discriminant function for class i with the value of $temp_{i-1}$ received from the previous stage.
4. Set the value of $temp\text{-}class_i$ based on the comparison in step 3.
5. Transfer the pixel data, $temp_i$ and $temp\text{-}class_i$ to the next stage.
6. If there is more data, go to step 1.
7. End.

The S-Net of stage i−1 of the pipeline is shown in Figure 3.1.8. A brief

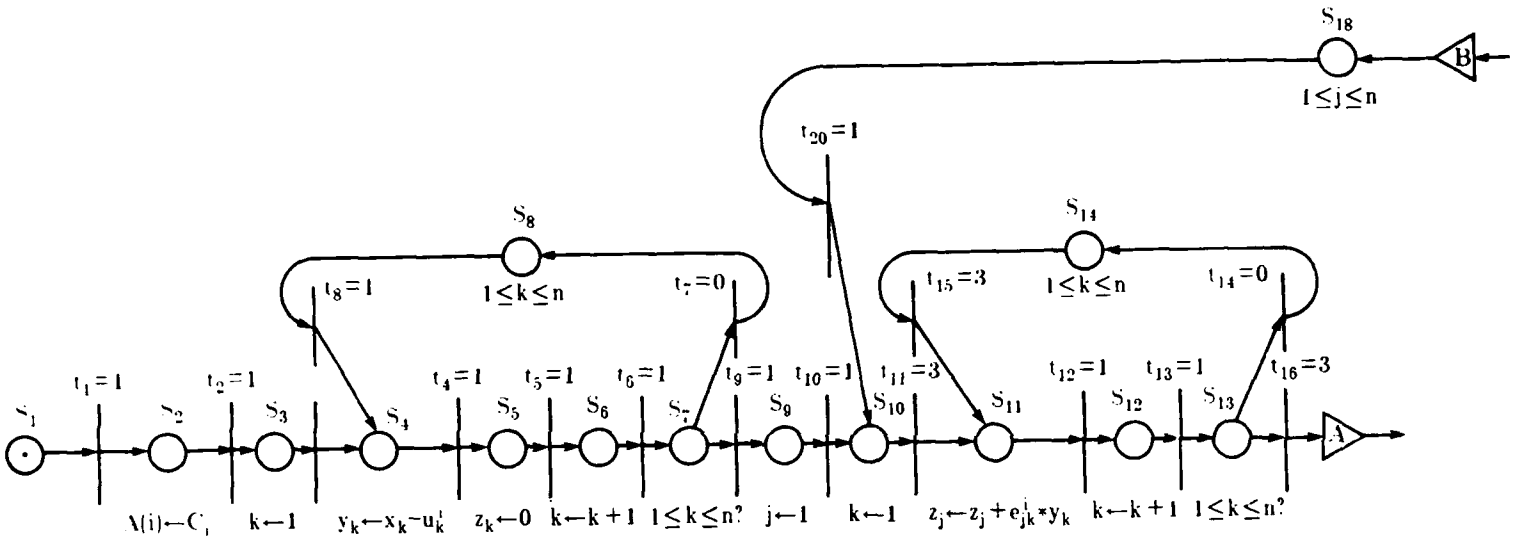


Fig. 3.1.8 Stage i-1 of the Pipeline

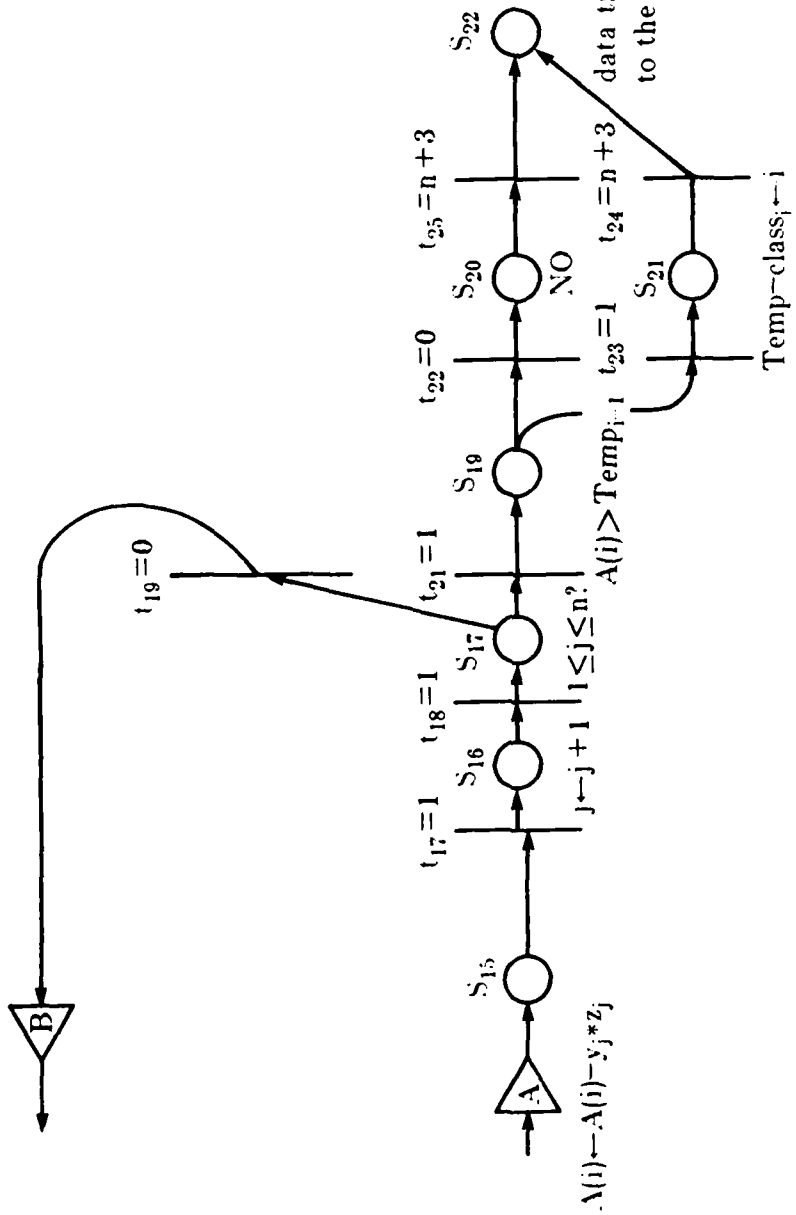


Fig. 3.1.8 (continued)

description for the S-Net is as follows:

- Figure 3.1.8 describes the computations of one processing element, so it has only scalar places.
- The initial marking of the S-Net is

  $K_0(S_1) = (1)$;

  $K_0(S_2) = K_0(S_3) = \ldots\ldots = K_0(S_{22}) = (0)$;

- Descriptors are:

  $D[t_1] = D[t_2] = \ldots\ldots = D[t_{25}] = [SFT;\_\_\ ]$;

- The token in S_1 indicates that this stage has received the 3-tuple data transfer from the previous stage. Once the data have been received, transition t_1 is enabled and fires.
- In most other respects, execution proceeds as in the SIMD case.

For calculating the degree of parallelism, we assume the S-Net goes through the transition t_{23} to t_{24}. E_{tot} is the total time units required to output one classified pixel. The timing analysis of the execution of the S-Net from t_1 to t_{24} is shown in Table 3.1.4. We have

$$E_{tot} = 5n^2 + 11n + 8 \text{ time units}$$

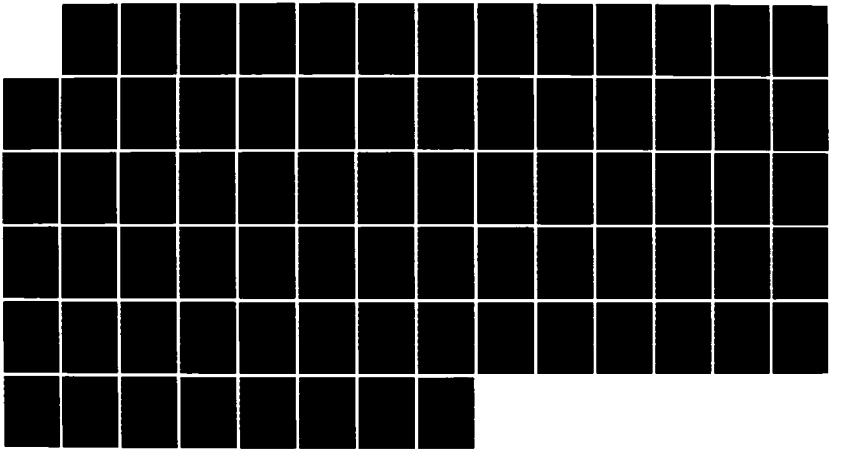
Since M PEs are active in parallel, the average parallelism $\bar{h}$ is (assume n = 4)

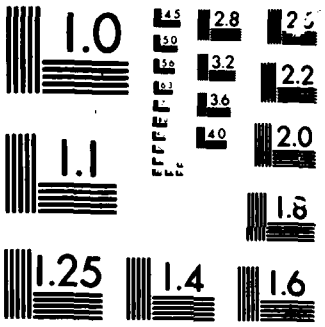
Table 3.1.4. Timing Analysis of Pipeline S-Net

transition t_i	no. of firings n_i	time the transition t_i takes, e_i	$n_i * e_i (* g_i)$
t_1	1	1	1(M)
t_2	1	1	1(M)
t_3	1	1	1(M)
t_4	n	1	n(M)
t_5	n	1	n(M)
t_6	n	1	n(M)
t_7	n−1	0	0(M)
t_8	n−1	1	n−1(M)
t_9	1	1	1(M)
t_{10}	1	1	1(M)
t_{11}	n	3	3n(M)
t_{12}	n^2	1	n^2(M)
t_{13}	n^2	1	n^2(M)
t_{14}	(n−1)n	0	0(M)
t_{15}	(n−1)n	3	3(n−1)n(M)
t_{16}	n	3	3n(M)
t_{17}	n	1	n(M)
t_{18}	n	1	n(M)
t_{19}	n−1	0	0(M)
t_{20}	n−1	1	n−1(M)
t_{21}	1	1	1(M)
t_{23}	1	1	1(M)
t_{24}	1	n+3	n+3(0)

AD-A167 317 DISTRIBUTED COMPUTING FOR SIGNAL PROCESSING: MODELING OF ASYNCHRONOUS PAR. (U) PURDUE UNIV LAFAYETTE IN SCHOOL OF ELECTRICAL ENGINEERING G LIN ET AL. AUG 84 2/2

UNCLASSIFIED TR-EE-84-29 ARO-18790.17-EL-APP-F F/G 9/2 NL





MICROCOPY CHART

$$\bar{h} = \frac{\sum_{i=1}^{24} n_i * e_i * g_i}{E_{tot}} = \frac{(5n^2+10n+5)M+(n+3)0}{5n^2+11n+8} = \frac{125\,M}{132}$$

The utilization of the M PEs is

$$\frac{\bar{h}}{M} = \frac{125}{132} = 0.947$$

Thus, the utilization of the 1024 PEs is 94.7%.

The total time units required to classify one image for the SIMD and pipeline architectures are as follows. For SIMD,

$$T_{SIMD} = [4 + 8m + 6mn + 3mn^2]\,M = 1.315 \times 10^6 \text{ time units}$$

For pipeline,

$$T_{pipeline} = [5n^2 + 11n + 8]mM = 2.163 \times 10^6 \text{ time units}$$

Since $T_{pipeline} > T_{SIMD}$, it has been shown that the pipeline requires more time to classify an image, even though the utilization of the PEs is greater. The reason, of course, is that for the pipeline the overhead resulting from the parallelism is greater. The pipeline could be made faster if the speed of the individual PEs could be increased through taking advantage of their much more specialized tasks and interconnections.

3.2. SIMD vs. MIMD

In the previous section, the description of SIMD processors is discussed using a graphic representation called S-Nets which is an extension of Petri nets. Several performance measures such as Average Vector Parallelism and Average Parallelism were described by which to evaluate the performance of SIMD processors.

In this section, the comparison between execution times of both SIMD and MIMD processors is discussed. When a particular algorithm such as the maximum likelihood classification algorithm or the clustering algorithm is given, it is necessary to ask which mode of parallelism, SIMD or MIMD, will have less execution time and better utilization of PE resources if the number of processing elements is fixed. In order to answer such questions, this section will analyze explicit and implicit operations [23] embedded in an algorithm and, based on these operations, derive the total execution times for both SIMD and MIMD modes of parallelism. In addition, the potential advantages and disadvantages, inherent in an algorithm, of MIMD mode of parallelism will be discussed as compared to SIMD mode of parallelism.

3.2.1. The Flow Chart and the Algorithms of the Supervised Relaxation Operator

The flow chart of the supervised relaxation operator is shown in Fig. 3.2.1. The algorithms in blocks A, B, and C can be executed simultaneously because the input data in these three blocks are independent of each other, and the algorithms are independent of results produced by each other. The algorithms can be implemented either in SIMD mode or in MIMD mode depending on the algorithm characteristics, which will be described in the following sections. Since the algorithm in block D needs the results from blocks A, B and C to execute its instructions, it is best that blocks A, B and C produce their results almost at the same time, then block D only needs to wait as little time as possible to proceed with its execution of instructions. The algorithms in blocks A, B, and C are quite different. Therefore their complexities in terms of execution time are also quite different. In order to produce the results almost

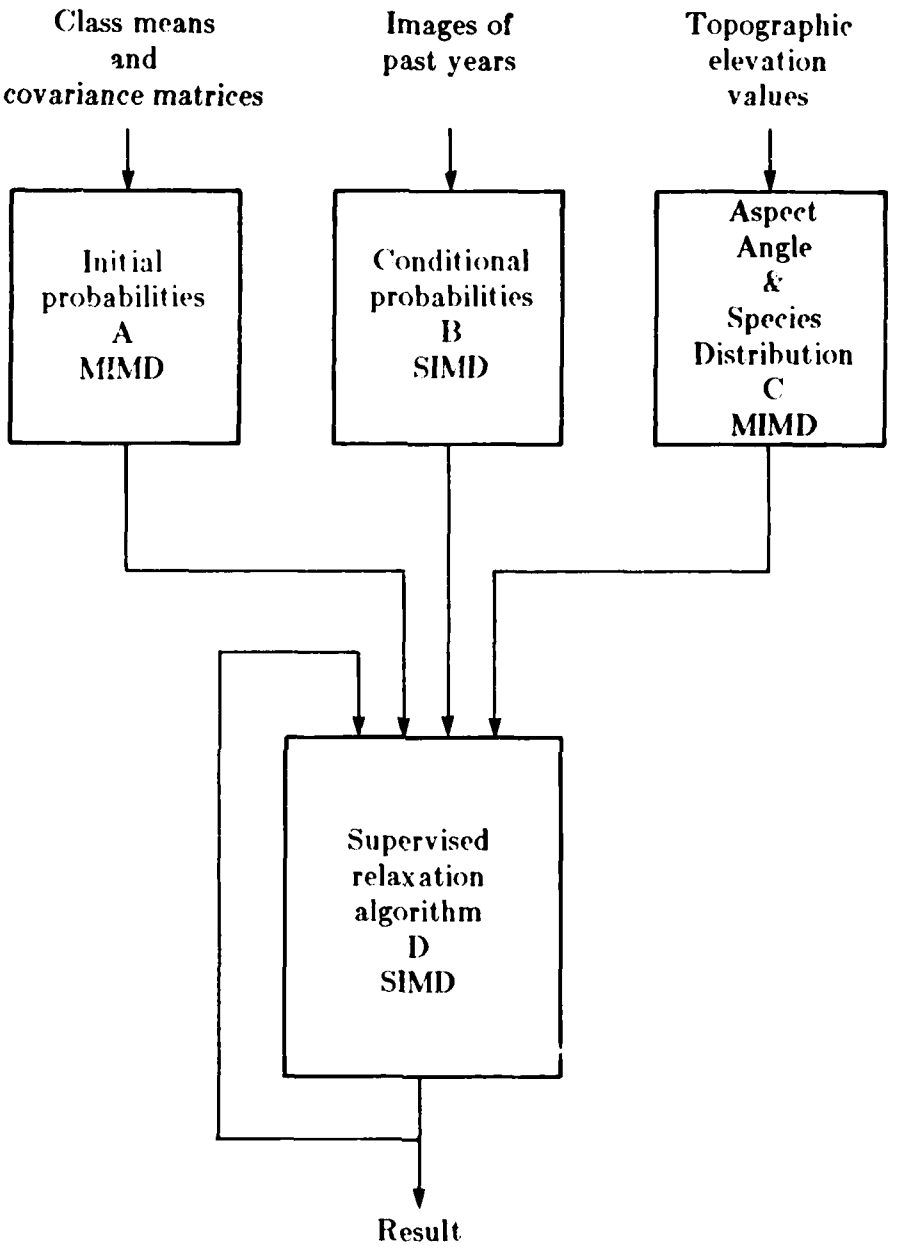


Fig. 3.2.1 The Flow Chart of the Supervised Relaxation Operator

simultaneously and provide better utilization of PEs, it is necessary to assign different numbers of PEs and different modes of parallelism to the implementation of these algorithms.

The algorithm in block A is actually the maximum likelihood classification algorithm except for the multiplication by a priori probability, $P(\lambda_i)$, for each class. If we assume that $P(\lambda_i) = 1/N$ where N is the number of classes, these two algorithms are exactly the same except for a constant factor. The algorithm in block A calculates the initial probability of every class for each pixel. Since this algorithm involves the multiplication of vector and matrix, matrix inversion, calculation of a matrix determinant, and an exponential operation, the execution time of each pixel depends on the multichannel data values. Especially for the exponential operation, its execution time is a function of argument value because it is a complex operation. An exponential operation is computed as a power series and the convergence of the power series depends on the argument value. In SIMD mode, the control unit broadcasts instructions to all active PEs and all active PEs execute the same instruction on the data in their own memories. Indeed all steps are synchronized. If this algorithm is implemented in SIMD mode, each PE executes the instruction on a different data value. As a result, each PE has a different execution time. Therefore, the PEs with shorter execution times have to wait until the PE with longest execution time completes its execution. Then the control unit can broadcast the next instruction to all PEs. The disadvantage of using SIMD mode is that the total execution time is equal to the summation of the maximum execution times of every operation because the execution time of the operation is data dependent. The advantages of using SIMD mode is that scalar instructions executed by the control unit and array

instructions executed by the PEs can be overlapped.

On the other hand, if the algorithm in block A is implemented in MIMD mode, each PE can execute an instruction fetched from its own memory immediately after completing the previous instruction. Therefore the total execution time is the summation of the execution times of every operation in the PE. If we assume that the data values are randomly distributed throughout the PEs, the probability of getting the data with longer execution time is the same for every PE. With this property, the total execution times of PEs in MIMD mode almost have the same values if the number of pixels in each PE is large enough.

The algorithm in block B involves only the accumulative operations of counting both individual and joint occurrences, and division operations. The execution times of these two operations are almost data independent. Therefore, no benefit can be drawn from implementing in MIMD mode. Furthermore, the algorithm has window-type operations and needs inter-PE transfers. SIMD mode can offer synchronized inter-PE transfers. That is, all active PEs can transfer data to the neighboring PEs located in the same direction relative to the transferring PEs. Since operations are simple and execution times are short, fewer PEs are assigned to block B when compared to block A in order for block A and B to take same amount of time to execute. SIMD mode is implemented to take advantage of synchronized inter-PE transfers and simple control system.

The algorithm in block C consists of two subalgorithms. One determines the aspect angle of the pixel. From this aspect angle, the other subalgorithm calculates the species distribution function of the classes which are possibly to be assigned to the pixel. The subalgorithm to determine the aspect angle has

window type operations and needs inter-PE transfers. The subalgorithm to calculate the distributing function is a neighbor-independent algorithm and involves complicated mathematical operations. If we assume that the distribution of each class is Gaussian and that the mean and variance of every class are already known and stored in the PE's memory, the relative likelihood of finding each class for the current pixel can be calculated plugging the elevation value into the Gaussian distribution function. As a result, more PEs are needed than in block B to speed up the process in order to synchronize the results produced from blocks A, B, and C and feed them to block D.

The algorithm in block D is the supervised relaxation operator. It involves operations such as addition, multiplication and division. We assume that these operations only span a narrow range of time if the argument value varies. Therefore this algorithm does not need to be implemented in MIMD mode. Instead it is implemented in SIMD mode. Every instruction is synchronized and the control system is simpler than that in MIMD mode.

3.2.2 Detailed Description of the Algorithm in Block C

The algorithm calculates the aspect angle [1] by numerically differentiating the topographic data to produce an estimate of the gradient vector at each pixel location. Then, the direction of the vector is used as aspect angle. The approximate gradient at line i and column j is computed as follows [1]:

$$\nabla\vec{\mathbf{Z}} \simeq \vec{\mathbf{I}}(Z_{i-1,j} - Z_{i+1,j}) + \vec{\mathbf{J}}(Z_{i,j-1} - Z_{i,j+1}) \qquad (3.2.1)$$

where

$\nabla\vec{Z}$ is the gradient vector

Z_{ij} is the topographic elevation value at i,j

i,j are line and column coordinates

$\vec{I}$ and $\vec{J}$ are line and column unit vectors

The aspect angle is calculated as follows [1]:

$$\alpha = \tan^{-1} \frac{(Z_{i-1,j} - Z_{i+1,j})}{(Z_{i,j-1} - Z_{i,j+1})} \qquad (3.2.2)$$

where

α is the direction of slope measured clockwise from north.

The subalgorithms for both aspect angle and species distribution are shown in Fig. 3.2.2. The subalgorithm for calculating the aspect angle of the current pixel decides that a pixel faces either south or north. Forest species distribution as a function of elevation and aspect in the San Juan Mountains is shown in Figure 2.1.4 [1]. In this figure, different aspects have different forest species distributions. That is the reason why the aspect angle is needed before calculating the distribution value. Constant C in Fig. 3.2.2 is a threshold to distinguish angle of south from that of north. In Fig. 3.2.2, the same subalgorithm is used to calculate the species distribution of south facing and north facing pixels. But the actual parameters such as m[k] and σ[k] which are the mean and the standard deviation of class k are different for south facing and north facing pixels. The constant C[k] in the subalgorithm is a precalculated value stored in the memory.

3.2.3 Implicit Operations in the Algorithm

Some of the statements in the following two sections are quoted from [23]. The conventional complexity analysis of an algorithm gives no knowledge about how fast an algorithm will execute for a given task of size N. Therefore this kind of analysis gives only the order of magnitude of the time that an

```
/*Algorithm for calculating aspect angle.*/
  for (i=0; i<n; i++)          /*subimage size is nxn */
    for (j=0; j<n; j++){
       V=(z[i-1][j]-z[i+1][j])/(z[i][j-1]-z[i][j+1]);
       α=tan^-1V;    /* α is the aspect angle */
       if (α > C)        /* pixel is south facing */
          calculate φ /*This is a subroutine which calculates species distribution*/
       else              /* pixel is north facing */
          calculate φ
    }

/* Subroutine to calculate species distribution function of each class for pixel
   of either south facing or north facing. */
/* φ(k) = C_k e^{(z - m_k^2)/(2σ_k^2)}  where C_k = 1/(√(2π) σ_k) */

   for (k=0; k<N; k++) {    /* N is the number of classes */
      temp = -(z[i][j]-m[k]) * (z[i][j]-m[k])/(2*σ[k] * σ[k]);
      φ[k] = C[k] * exp(temp);    /* φ[k] is the distribution value for class k. */
   }
```

Fig. 3.2.2. The Algorithm for Aspect Angle and Species Distribution

algorithm will take for a given size N, such as "order N" or "order N^2." Several papers have given the analysis of execution time by counting the number of explicit operations that an algorithm will perform [24,25]. However, there are many implicit operations neglected in the analysis such as the calculation of the real address of index variable z[i][j], moving operands from memory into machine registers, and moving results from machine registers back into memory [1]. Counting only explicit operations gives only an analysis of explicit arithmetic operations, and not the implicit operations. Often these implicit operations can have a significant impact on the execution time of an algorithm and therefore should not be ignored.

In Fig. 3.2.2, there are many indexed variables such as z[i][j], m[k] and σ[k]. Let us look one example to identify implicit operations. For example, in order to get the value of m[k] into a machine register, the base address of array m must be found, put into a machine register, added to i, and the resulting address used to load the value of m[k]. As another example, loading the value of z[i][j] into a machine register requires that i be multiplied by the row size of the z array, added to j, the result added to base address of the z array, and the resulting address used to load the value of z[i][j]. Alternatively, to avoid the multiplication, an indirection table could be used. The method requires that i be added to the base address of the indirection table, the resulting address used to load the address of the ith row of array z, which is added to j to get the

address of z[i][j]. Finally, this address is used to load the data itself. While this method saves a multiplication, it requires an additional memory load and space for the indirection table.

The notation M[address] will be used to denote a memory reference to address. The implicit operations required to access operands are summarized in Table 3.2.1.

Before counting, in Section 3.2.4, the total number of implicit and explicit operations in the program shown in Fig. 3.2.2, several assumptions [23] are listed as follows:

1. The class means and standard deviations, m[k] and σ[k], k = 1,2,...,N, have been stored in each PE.
2. C[k] = $1/(\sqrt{2\pi}\sigma[k])$, k = 1,...,N, have been precalculated and stored in each PE.
3. There are not enough machine registers to hold all data of the program shown in Fig. 3.3.2.
4. There are enough machine data registers to hold all index variables currently in use such as i, j, and k in the program.
5. There are enough machine data registers to hold all temporary results such as V and temp, and repeated expressions such as σ[k] and a compiler is capable of recognizing and exploiting this fact. That is, the compiler will not generate store and re-load operations in these situations.
6. There are enough address registers to hold the addresses of all single variables such as V and α, and the base addresses of array variables such as z, m, and σ.

Table 3.2.1. Implicit Operations to Access Operands [23]

To Access	Notation	Operations
c	M[c]	1 memory reference
m[k]	M[base of m + k]	1 addition 1 memory reference
z[i][j]	M[M[base of z + i] + j]	2 additions 2 memory references

7. Variables already in machine registers have no implicit operations associated with them.

8. Implicit operations include the movement of constant data and new addresses from the instruction stream into data and address registers. When variables are first used, their addresses are migrated from the instruction stream to address registers. Addresses and data in registers are discarded when no longer needed.

Usually, different algorithms spend a different portion of run time executing implicit operations. Therefore the relative times needed to perform various types of operations are important. For many processors, multiplication and division operations take from 10 to 50 times as long as addition or subtraction operations. Thus if a program contains many explicit *multiplication* and division operations, the effect of the implicit operations may not be significant. On the other hand, many algorithms contain no multiplication and division operations at all. Under these circumstances, explicit operations alone give a very poor estimate of the total algorithm execution time. Realizing the importance of the relative times of various operations, a list of times, in cycles, for various operations is given in Table 3.2.2 [23]. The times given are for the Motorola MC68000, a typical modern multiregister processor. The internal cycle time for an 8MHz MC68000 is 250 ns. All of the figures given are in "internal cycles."

Table 3.2.2. Cycle Times of Various Operators [23]

OPERAND	CYCLES	SIZE
(operands in registers)		
addition, subtraction, (A,S)	2	word
	3	long
boolean, comparison	2	word
	3	long
shift	3+shiftcount	word
multiplication (M)	35	word*word=long
division (D)	75	long/word=word
(address in a register)		
load, store (R)	4	word
	6	long
(immediate)		
load immediate address or constant data	4	word
	6	long
(operand in a register)		
test and branch on condition	5	
(address in a register)		
subroutine call	9	
save/restore n registers on the stack	4+4n	
subroutine return	9	
interrupt	21	

3.2.4. Comparison Between SIMD and MIMD

Some assumptions of the processing environments of SIMD and MIMD [23] are repeated here. For SIMD mode:

1. The control unit broadcasts the instruction stream to all active PEs and all active PEs execute the same instruction simultaneously on the data in their own memory. While active PEs execute the instructions in parallel, the control unit can do scalar instructions such as loop counting, branching, and subroutine calls and returns. These control unit operations are overlapped with PE operations and thus do not contribute to the overall algorithm execution time. Since mask operations are decoded in the control unit, PE address and general masking operations cost nothing in the total algorithm run time.
2. *Data transfers* through the interconnection network are performed simultaneously. The network transmission delay for a circuit-switched multistage network is less than 2 cycles. The total cost of an inter-PE transfer is $4 + 2 + 4(= 10)$ cycles for the load-transfer-store sequence.
3. "If any," "if all" or "where" conditional steps require about 25 cycles. Control unit and PE operations cannot be overlapped for these statements.

For MIMD mode:

1. PEs fetch instructions from their own memory and do all computational and control (branching) operations.
2. The data transfers through the interconnection network operate asynchronously. Each transfer causes an interrupt at the destination

PE. The time to service the interrupt includes the time to save registers, call the interrupt service routine, load the incoming data from the I/O port, store the data in an appropriate buffer, and return to the interrupted routine.

The algorithm to calculate the aspect angle of the pixel includes window type operations and therefore requires inter-PE data transfers. The image to be processed is superimposed on the PEs, which are arranged in a mesh-type array. The image is divided among the PEs and each PE is responsible for the corresponding subimage. The window-type operation needs data from one or two neighboring PEs when it processes any border pixel of its own subimage. In SIMD mode, these data transfers are performed simultaneously for all PEs via the interconnection network. Specifically, for an n×n subimage, a total of 4n parallel transfers are needed for the algorithm to calculate an aspect angle. From assumption 2 for SIMD mode, each parallel transfer costs 10 cycles to complete the operation. On the other hand, in MIMD mode, each data transfer causes an interrupt at the destination PE and this operation is carried out asynchronously by the PE. Assuming the same size subimage, each PE causes 4n interrupts at the destination PE, and gets interrupted 4n times by other PEs when data are ready to be transferred. Therefore, each PE totally has 8n interrupts for MIMD mode. Deadlock is the situation which occurs when two PEs interrupt each other and each waits for the other PE to continue. Deadlock can be prevented if the procedures below are followed [26]:

1. If a PE gets interrupted, then no other PEs can interrupt this PE.
2. If the same PE gets interrupted simultaneously by more than one PE, one of them is given a higher priority.

3. If PEi and PEj interrupt each other, then each PE has the capability to grant one of the interrupts and disable the other. For example, PEi has the capability to detect that it interrupted PEj and that PEj interrupted PEi. Therefore, if PEi has higher priority, PEj will grant PEi's interrupt request and PEj will withdraw its interrupt to PEi. A similar situation occurs if PEi interrupts PEj and PEi gets interrupted simultaneously by PEk.

Since the loop counting operations in SIMD mode are executed by the CU and overlapped with array instructions executed by PEs, they cost nothing in the total algorithm run time. On the other hand, loop counting operations in MIMD mode are executed by the PEs and not overlapped with array instructions. For example, the "C" program instruction in Fig. 3.2.2 •

$$\text{for } (i = 0;\ i < n;\ i + +)$$

requires n additions and $n + 1$ comparisons. These operations are in the category of loop counting operations.

There is no memory contention problem in SIMD mode since the elevation values z[i][j] of the $n \times n$ subimage and all the parameters such as m[k] and σ[k] are stored in the corresponding memory of the PEs. PEs only fetch data from their own memories. However, there are two alternative memory organizations possible in MIMD mode. One is the MIMD mode with global memory which stores the parameters such as m[k] and σ[k], but the elevation values for the subimage in each PE are stored in local PE memory. Serious memory contention will occur when every PE tries to access m[k] and σ[k] to calculate the frequency responses of all classes. The other MIMD mode uses only local memory. In this case, all parameters and data are stored in local memory. Of course, it requires more total memory. All PEs can access the data from their

own memories without memory contention. Here, we consider the MIMD mode with local memory only.

The algorithm shown in Fig. 3.2.2 requires (2 + N) subtractions, (1 + N) divisions, 4N multiplications, 1 comparison, 1 conditional step, 1 arctangent, and N exponential operations per pixel where N is the number of classes. These operations are in the category of explicit operations.

Now we are in a position to enumerate the implicit operations associated with the program shown in Fig. 3.2.2. It is found that the algorithm requires 1 operation to move the constant 2 to a data register, 9 operations to move the addresses of variables such as V, z, α... etc. to address registers, (4N+14) additions, and (4N+10) memory references per pixel. A memory reference is either a memory store or a memory load operation.

The number of operations involved in executing the algorithm in SIMD, MIMD, and serial modes are listed Table 3.2.3. The operations to move data and addresses into data and address registers are included in immediate operations. In SIMD mode, data transfers are via interconnection networks and not via interrupt operations. Loop counting operations are overlapped with PE instructions. The same situation doesn't occur in either MIMD or serial processor modes. In MIMD mode, data transfers are executed on demand via interrupt operations but not through parallel transfers. In serial processor mode, the overhead of parallelism, present in both SIMD and MIMD modes, doesn't exist.

In Table 3.2.3, the execution times of most operations are argument-independent except two: the arctangent and exponential operations. In SIMD mode, since every instruction is a lockstep operation, the PEs with shorter instruction execution time has to wait for completion of the PE with longest

Table 3.2.3. Operations in SIMD, MIMD, and Serial Processor

	SIMD	MIMD	SERIAL
Interrupt	0	8n	0
Parallel Transfer	4n	0	0
Loop Counting:			
addition	0	$n^2N+n(n+1)$	M^2N+M^2+M
comparison	0	$n^2(N+1)+(n+1)^2$	$M^2(N+1)+(M+1)^2$
Implicit Operation:			
addition	$n^2(4N+14)$	$n^2(4N+14)$	$M^2(4N+14)$
memory reference	$n^2(4N+10)$	$n^2(4N+10)$	$M^2(4N+10)$
immediate	10	10	10
Explicit Operation:			
subtraction	$n^2(2+N)$	$n^2(2+N)$	$M^2(2+N)$
division	$n^2(1+N)$	$n^2(1+N)$	$M^2(1+N)$
multiplication	$n^2(4N)$	$n^2(4N)$	$M^2(4N)$
comparison	n^2	n^2	M^2
condition	n^2	n^2	M^2
arctangent	n^2	n^2	M^2
exponential	n^2N	n^2N	M^2N

M x M = image size

n x n = subimage size

N = number of classes

instruction execution time. Therefore, the total algorithm run time is the summation of the maximum execution times of every operation. The lockstep operation mode will thereby lead to inefficient utilization of PEs in SIMD. On the other hand, PEs in MIMD mode can execute the next instruction immediately after completing the current instruction. Therefore, parallelism in MIMD mode has the potential to save execution time. The cycle times of various operations are listed in Table 3.2.2. But there is no data available from which to estimate the range of the execution times of either arctangent or exponential operations. For most processors, multiplication and division operations take from 10 to 50 times as long as addition or subtraction operations. Exponential and arctangent operations are complex operations which involve many multiplication and division operations. Therefore, a reasonable range of execution times of these two operations are assumed in order to carry out the comparison of the execution times of SIMD and MIMD modes. Further, it will be assumed that the argument values are randomly distributed throughout the whole image so that the algorithm run time is approximately equal to the sum of the means of every operation execution time in each PE.

The plane of execution times for SIMD mode is shown in Fig. 3.2.3 under the assumption that the range of maximum execution times of both operations is from 200 to 400 cycles. Other assumptions used to calculate the algorithm execution time are that an interrupt costs 21 cycles, a parallel transfer 10 cycles, an addition 2 cycles, a comparison 2 cycles, a memory reference 4 cycles, an immediate operand 4 cycles, a subtraction 2 cycles, a division 75 cycles, a multiplication 35 cycles, a conditional step 25 cycles; the subimage size is $n = 32$, the number of classes $N = 5$, and the whole image size $M = 256$. The z-

axis is execution time, the x-axis is the range of the execution time of the arctangent operator, and the y-axis is the range of the execution time of the exponential operator. The range of the mean instruction execution time is assumed to be from 180 to 360 cycles for both operators. The plane of the execution time for MIMD mode is also shown in Fig. 3.2.3. If the range of the execution time is different from what we have assumed here, the results are expected to be similar. The results shown in Fig. 3.2.3 only indicate the trend of the difference of the execution times between SIMD and MIMD modes. This figure shows that MIMD mode has less execution time than SIMD mode. Although scalar operations can not overlap with array operations in MIMD mode, the execution time saved due to efficient utilization of PEs in MIMD mode leads to less execution time and better utilization of PEs in MIMD mode.

3.3 Alternative Performance Criteria

Feature enhancement methods, commonly used in producing maps from imagery data, enhance or emphasize information-bearing attributes of the data while, ideally, suppressing other "noise" characteristics. The distinction between "information" and "noise" is highly application-dependent. For the purpose of this study, attention is focused on *spectral similarity*, an image attribute which is commonly found to be useful in scene analysis. If the imagery is black-and-white, spectral similarity reduces simply to tonal (gray-scale) similarity. A *clustering algorithm* will be described, including a parallel implementation, which can be used to identify sets of spectrally similar pixels. The pixels need not be spatially contiguous, but simply have the same "color" in a generalized sense. This algorithm will be used to illustrate alternative performance criteria for evaluating parallel (SIMD) algorithms.

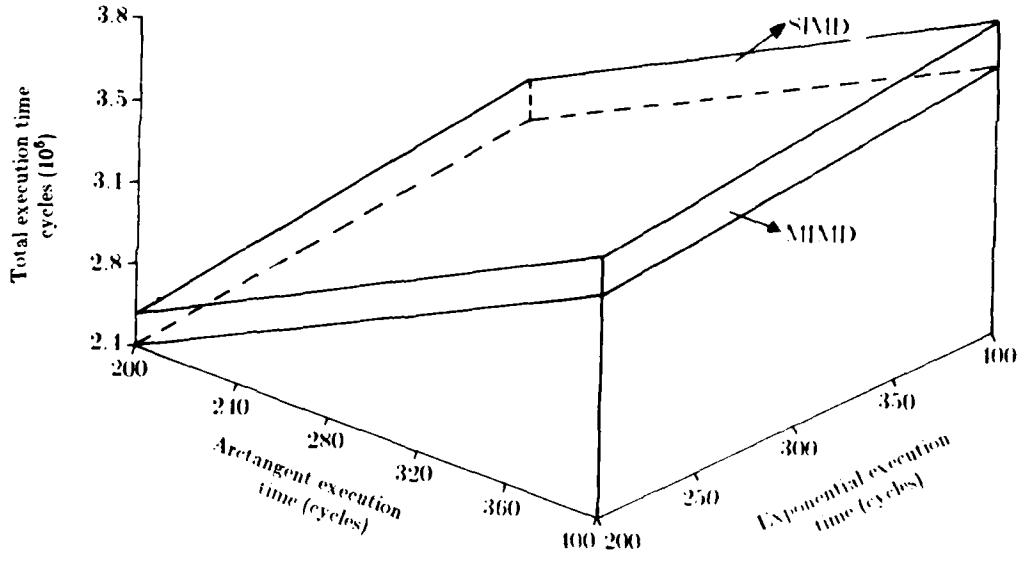


Fig. 3.2.3. Planes of Execution Time for SIMD and MIMD

In general, the complexity of SIMD algorithms is a function of the problem size (number of elements in the data set to be processed), machine size (number of PEs), and the interconnection network used to provide communications among the PEs. For example, an algorithm which uses N PEs to execute some operation on an M × M image will exhibit different "performance" for different values of N. The obvious use of performance measures is for selecting from alternative SIMD algorithms. For a given SIMD machine, different algorithms for performing a particular task can be compared. The algorithm which performs best based on the desired measurement criteria can then be chosen. As another use for performance measures, consider the situation where the typical values of image size M are known. Then a measure of the way in which the machine size affects the performance of the application algorithms will be useful in deciding how many PE's the system should have. Lastly, given a reconfigurable system [27], the machine size can be tailored to the problem size for execution of a given algorithm if there exists performance criteria for comparing different choices of machine size. The goal of this section is to study the relationships among the various parallel configurations. In order to demonstrate one way in which the measures can be applied, an SIMD clustering algorithm is presented. In this example algorithm evaluation, both the image size and the machine size are varied, permitting the performance of the algorithm to be examined and compared under a variety of conditions.

3.3.1 A Clustering Algorithm

Given n-dimensional vectors $x_a = [a_1 a_2, \ldots, a_n]^T$ and $x_b = [b_1 b_2, \ldots, b_n]^T$, the Euclidean distance between the vectors is defined by

$$D = \left[\sum_{i=1}^{n} (a_i - b_i)^2\right]^{1/2}$$

Given a population of *n*-dimensional vectors normally distributed $N(U_i, \Sigma_i)$ and a second population normally distributed $N(U_j, \Sigma_j)$, where U_i, U_j are the population means and Σ_i, Σ_j are the population covariance matrices, the distance between these populations is defined to be the *divergence*

$$D_{ij} = \frac{1}{2} \text{ tr } \left[(\Sigma_i - \Sigma_j)(\Sigma_j^{-1} - \Sigma_i^{-1})\right]$$

$$+ \frac{1}{2} \text{ tr } \left[(\Sigma_i^{-1} + \Sigma_j^{-1})(U_i - U_j)(U_i - U_j)^T\right]$$

The following clustering algorithm is based on the ISODATA algorithm of Hall and Ball [28]. It groups vectors in such a way as to minimize the sum-of-squared-error (SSE):

$$SSE = \sum_{i=1}^{c} \sum_{x \in c_i} \|x - M_i\|^2$$

where

c = number of clusters

c_i = the set of vectors belonging to the ith cluster

M_i = the mean vector for the ith cluster

Intuitively the vectors are grouped as tightly as possible about their respective cluster means.

Step 1: Select c arbitrary but distinct vectors to serve as initial cluster centers, $\dot{M}_i$, i=1,2,...,c. (The user specifies the value of c.)

Step 2: Assign each vector in the data set to the nearest cluster center, based on Euclidean distance.

Step 3: Compute the mean vectors for the data assigned to each cluster. Denote means by M_i, i=1,2,...,c.

Step 4: If the new cluster means M_i are identical with the old cluster centers $\dot{M}_i$, go to Step 5. Otherwise set $\dot{M}_i = M_i$, i=1,2,...,c and return to Step 2.

Step 5: Compute the intercluster distances (divergences) and merge indistinct clusters (having divergences less than a prespecified threshold). Clustering complete.

The clustering algorithm is depicted in Figure 3.3.1. On every iteration of steps 2, 3, and 4, the reassignment and movement of cluster centers (means) reduces the SSE. Since there is a lower limit on the SSE (it cannot be made less than zero), the algorithm is guaranteed to terminate via step 5.

3.3.2 Parallel Implementation

Figure 3.3.2 contains a parallel implementation of the clustering algorithm in a high-level programming language. The implementation is of the SIMD type. N is the number of PEs and $M \times M$ is the image size. The PEs are configured logically as a $\sqrt{N}$-by-$\sqrt{N}$ grid on which the M-by-M image is superimposed, so that each PE holds a $\frac{M}{\sqrt{N}}$-by-$\frac{M}{\sqrt{N}}$ subimage (for convenience, it is assumed that $M/\sqrt{N}$ is an integer). The "local" assignment

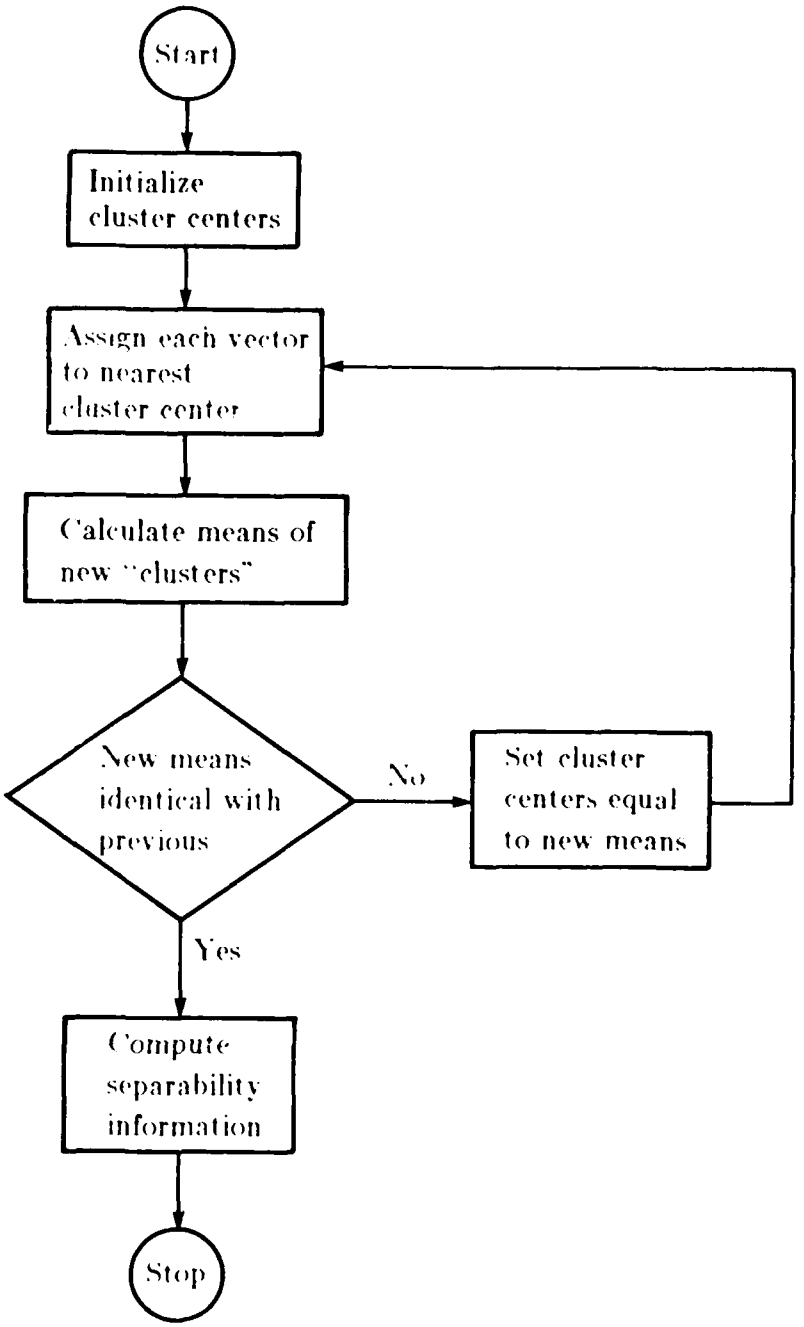


Fig. 3.3.1. A Basic Clustering Algorithm

```
/* Iterative Clustering Algorithm */

/* All PEs execute in parallel.
   number(i), i=1,....c stores the number of pixels in the corresponding class.
   Assume all initial c cluster centers are already stored in each PE.
   Let M̂_i = [m̂_i1,m̂_i2,....m̂_in]^T, i=1,2,....c be the c initial cluster centers.
   Let M_i = [m_i1,m_i2,....,m_in]^T, i=1,2,....c be the new cluster means.
   n is the number of features per pixel. N is the number of PEs and
     the image size is MxM
   Assume this is a checkerboard type data allocation scheme.
   PEs are arranged in a √N-by-√N array and each PE
     stores a M/√N-by-M/√N subimage. */

/* Initialization. Zero the means and pixel counts. */

for i ← 1 to c
   do for j ← 1 to n
         do m_ij ← 0
      end
      number (i) ← 0
end

/* Nearest neighbor assignment */

/* Let x be the spectral measurement vector of a pixel where x = [x_1,....,x_n]^T and
   n is the no. of features per pixel.
   Array pclass stores the cluster no. to which pixels belong.
   m_i,j = mean for class i, feature j at end of iteration
   pchange stores the no. of pixels which change class
     assignment from last iteration to this iteration */
```

Fig. 3.3.2. Parallel Implementation of the Clustering Algorithm (continued on following pages)

```
pchange ← 0
for k ← 0 to M/√N − 1
    do  for ℓ ← 0 to M/√N − 1
            do distance ← 10000000
                for i ← 1 to c
                    do eudist ← 0
                        for j ← 1 to n
                            do eudist ← eudist + (x_j−m_ij)*(x_j−m_ij)
                        end
                        if eudist < distance then class ← i
                                                  distance ← eudist
                end

                for i ← 1 to n
                    do m_class,i ← m_class,i + x_i
                end

                number (class) ← number (class) + 1
                if (pclass (k,ℓ).NE. class) pchange ← pchange +1
                pclass (k,ℓ) ← class
        end
end

/* Use recursive doubling algorithm to merge the elements in vectors M_i, i=1,...,c
     and array number (i), i=1,...,c from each PE.
   ṅ = log_2N
   After merging, the results will be stored in each PE. */

for i ← 0 to ṅ − 1
    do
        DTR_in ← DATA to be added
        TRANSFER using Cube_i
        DATA ← DATA + DTR_out
end
```

Fig. 3.3.2. (continued)

```
/* Compute the new cluster means, M_i, i=1,...,c */

for i ← 1 to c
    do  for j ← 1 to n
            do m_ij ← m_ij/number (i)
        end
end

/* Compute the percentage of class change for the whole image */

/* Use recursive doubling algorithm to merge pchange in each PE.
   After merging, the result will be stored in PE 0. */

for i ← 0 to n - 1
    do  MASK [X^(n-i-1) 1 X^i]
        DTR_in ← pchange
        TRANSFER using Cube_i
        MASK [X^(n-i-1) 0 X^i]
        pchange ← pchange + DTR_out
end
MASK[0^n]
pchange ← (pchange/(M*M)) * 100

/* If pchange is less than a prespecified threshold, then the clustering
   is complete. Compute the separability information (not shown here);
   otherwise set M'_i = M_i, i=1,2,...,c, and
   return to the beginning of this algorithm. */
/* To test if pchange is less than t or not, only PE 0 is enabled. */

if pchange < t  then compute the separability
                else M'_i ← M_i, i=1,2,...,c
                     go to the beginning of this algorithm.
```

Fig. 3.3.2. (continued)

of pixels to cluster centers is performed in parallel, so that each PE contains the result of a clustering iteration for the subimage which it holds. To compute the new cluster means, these local results are then combined using a form of overlapped recursive doubling.

The exact data allocation scheme used for this algorithm is not critical. The only requirement is that 1/N of the image elements be assigned to each PE.

For simplicity, we shall assume a scenario with monochrome imagery, so that the number of features per pixel (n) is 1 for this algorithm. Then on each iteration, the distance calculation and determination of cluster membership requires (3 additions + c subtractions + (c+1) comparisons + c multiplications) $* \frac{M^2}{N}$ operations for each PE. The recursive doubling algorithm to merge the means M_i, and constants number(i), i=1,...,c in each PE requires 2c parallel transfers and add operations at each step. Since there are $\log_2 N$ steps to merge the data, a total of $2c\log_2 N$ parallel transfers and additions is required. After merging the data, c division operations are needed to compute the new cluster centers. Using the recursive doubling algorithm to merge pchange from each PE requires $\log_2 N$ transfers and additions, and, $2\log_2 N$ masking operations. Computing the percentage of class change requires (2 multiplications + 1 division) operations and 1 mask operation. Finally, to test if pchange is less than a prespecified threshold t requires 1 comparison; only PE 0 is enabled at this step.

3.3.3 Performance Analysis

For application of the performance measures to the clustering algorithm, the following are assumed:

t_a = time for 1 integer addition operation,

t_s = time for 1 integer subtraction operation,

t_m = time for 1 masking operation,

t_c = time for 1 integer comparison operation,

t_y = time for 1 integer multiplication operation,

t_d = time for 1 integer division operation,

$t_t(N)$ = time for 1 parallel data transfer operation.

For simplicity in the example that follows, it will also be assumed that $t_a = t_s = t_c = t_y/2 = t_d/2 = 2t_m = t_t(N)/2$. The actual relationships among these operations will be implementation dependent. Time for loading and storing data items and for program control operations such as testing loop indices has not been explicitly included in the analysis. The time required for program control will be assumed negligible in comparison to the other times, and, in general, the program control operations can be overlapped with the PE and inter-PE transfer operations. As shown earlier, the analysis could be modified to account for implicit operations.

The performance of the clustering algorithm will be evaluated as a function of N for an M-by-M image, with M ranging from $2^6 = 64$ to $2^{13} = 8192$. The evaluation is limited to machines having between 2^6 and 2^{14} PEs and the number of clusters being 32.

The sequential algorithm to do the clustering requires M^2 [3 additions + c subtractions + (c+1) comparisons + c multiplications] + c divisions + 1 comparison. Thus, the serial execution time (one processor, M^2 pixels) is given by $T_1(M^2) = M^2$ [3 additions + c subtractions + (c+1) comparisons + c multiplications] + c divisions + 1 comparison = $M^2 [3*t_a + c*t_s + (c+1)*t_c + c*t_y] + c*t_d + t_c$.

The following performance criteria applied to the clustering algorithm are discussed in [29].

(a) Execution Time (N processors, M^2 pixels):

$$T_N(M^2) = \frac{M^2}{N} [3*t_a + c*t_s + (c+1)*t_c + c*t_y] + (c+1)*t_d + t_c + (2c+1)\log_2 N*t_a + 2*t_y + (2c+1)\log_2 N*t_t(N) + (1+2\log_2 N)*t_m$$

The execution time can be expressed as the sum of two components, computation time and overhead due to parallelism. The computation time is given by

$$c_N(M^2) = \frac{M^2}{N} [3*t_a + c*t_s + (c+1)*t_c + c*t_y] + (c+1)*t_d + t_c + (2c+1)\log_2 N*t_a + 2*t_y$$

The overhead is given by

$$O_N(M^2) = (2c+1)\log_2 N*t_t(N) + (1+2\log_2 N)*t_m$$

The serial execution time is $T_1(M^2)$. Figure 3.3.3 shows the $\log_2$ of the execution time as a function of N and M under the simplifying assumptions outlined above. The graph has been normalized to $t_a = 1$. For large images (e.g., $M \geq 1024$), it is clear that for given M the execution time decreases as N increases. For such M, if N is doubled, then the execution time is decreased by

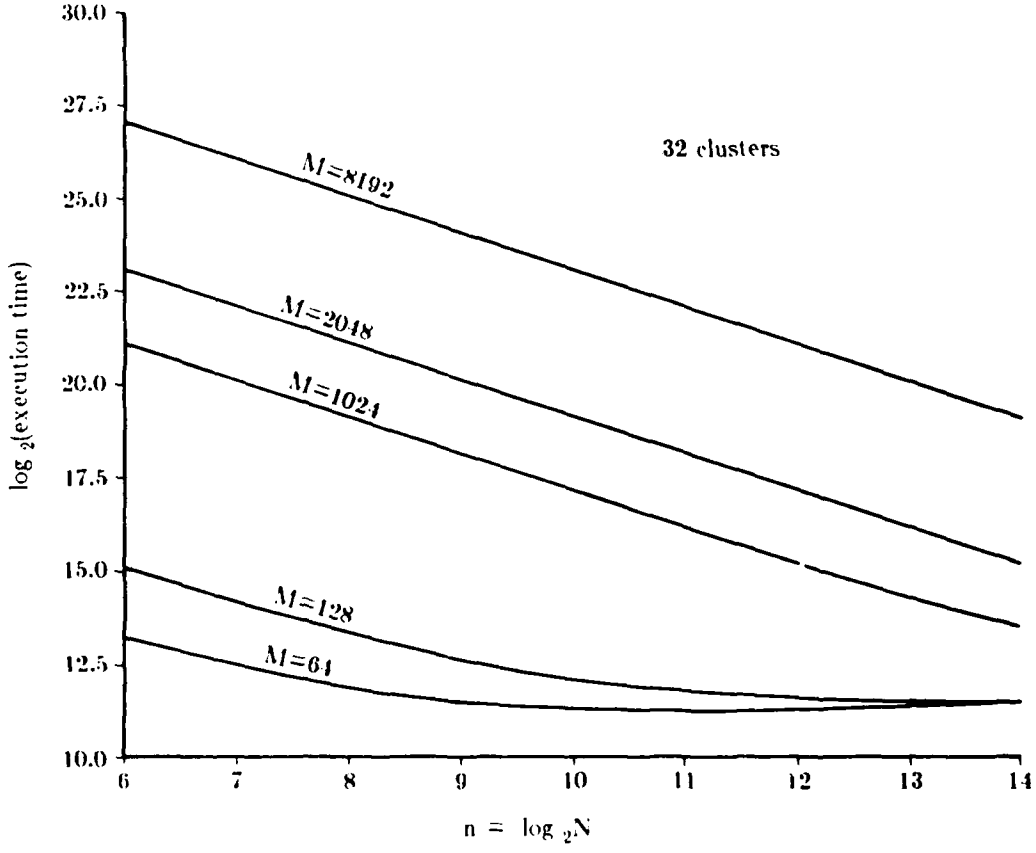


Fig. 3.3.3. Clustering Algorithm: Execution Time

approximately a factor of two. For small images, an increase in the number of PEs has little effect on the execution time. For such M, the time required for the recursive doubling algorithm dominates when N is large, and there is little or no advantage to using a large machine. For each M in the range 64 to 128, the rate at which $T_N(M^2)$ decreases ($dT_N(M^2)/dN$) also decreases as N increases. Therefore, for practical purposes it may be appropriate to consider $dT_N(M^2)/dN$ as well as $T_N(M^2)$.

(b) Speed-Up:

$$S_N(M^2) = T_1(M^2)/T_N(M^2)$$

Figure 3.3.4 shows the $\log_2$ of the speed-up as a function of N and M, under the simplifying assumptions. For large M, $S_N(M^2) \simeq N$, i.e., the speed-up is almost ideal for all N considered. Therefore, using $S_N(M^2)$ as the performance criterion would dictate using as many PEs as are available. For small M, $S_N(M^2) \ll N$, and the choice of N has little effect on the speed-up. For example, for M = 64, there is little advantage to using more than $2^9 = 512$ PEs. For small M, there exists a value for N up to which increasing the number of PEs significantly increases the speed-up, but beyond which there is little advantage to increasing N. Thus, it appears that using a combination of speed-up with d(speed-up)/dN (and/or a measure such as utilization or efficiency - see below) is a more practical criterion than speed-up alone.

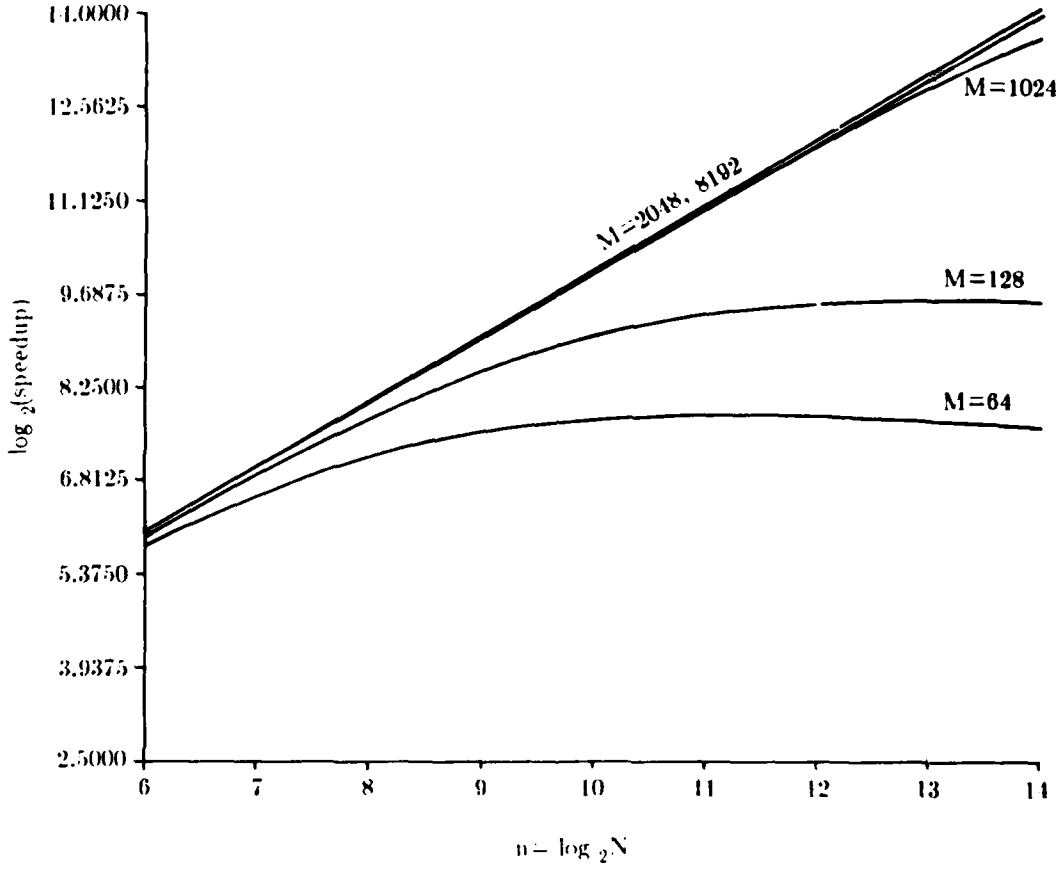


Fig. 3.3.4. Clustering Algorithm: Speedup

(c) Efficiency:

$$E_N(M^2) = S_N(M^2)/N$$

$$= T_1(M^2)/(N*T_N(M^2))$$

Figure 3.3.5 shows the efficiency for a range of values of M, under the simplifying assumptions. As required by the definition, $0 \leq E_N(M^2) \leq 1$. For all M, the efficiency decreases as N increases, but the rate of decrease is slower for large M. For large M, the efficiency is high for all N considered, and the choice of N does not significantly affect $E_N(M^2)$. For small M, $E_N(M^2)$ is small for large N. The conclusion that the efficiency is poor for the choices of large N and small M is consistent with the information from the execution time measure. From the observation that $E_N(M^2)$ is higher for large M, it can be concluded that for a fixed N, there may be no advantage to decomposing a size MxM problem into smaller subproblems, even if the result can later be recombined at low cost.

(d) Utilization:

$$U_N(M^2) = \sum_{x=0}^{X-1} t_x P_x/(N*T_N(M^2))$$

where t_x = time perform step x

P_x = no. of PEs active for step x

X = no. of steps in the N PE computation

(The computation time $c_N(M^2) = \sum_{x=0}^{X-1} t_x$).

To derive the utilization requires counting the number of PEs active for each computation step. In the stage of merging the pchange in each PE, $\log_2 N$ steps

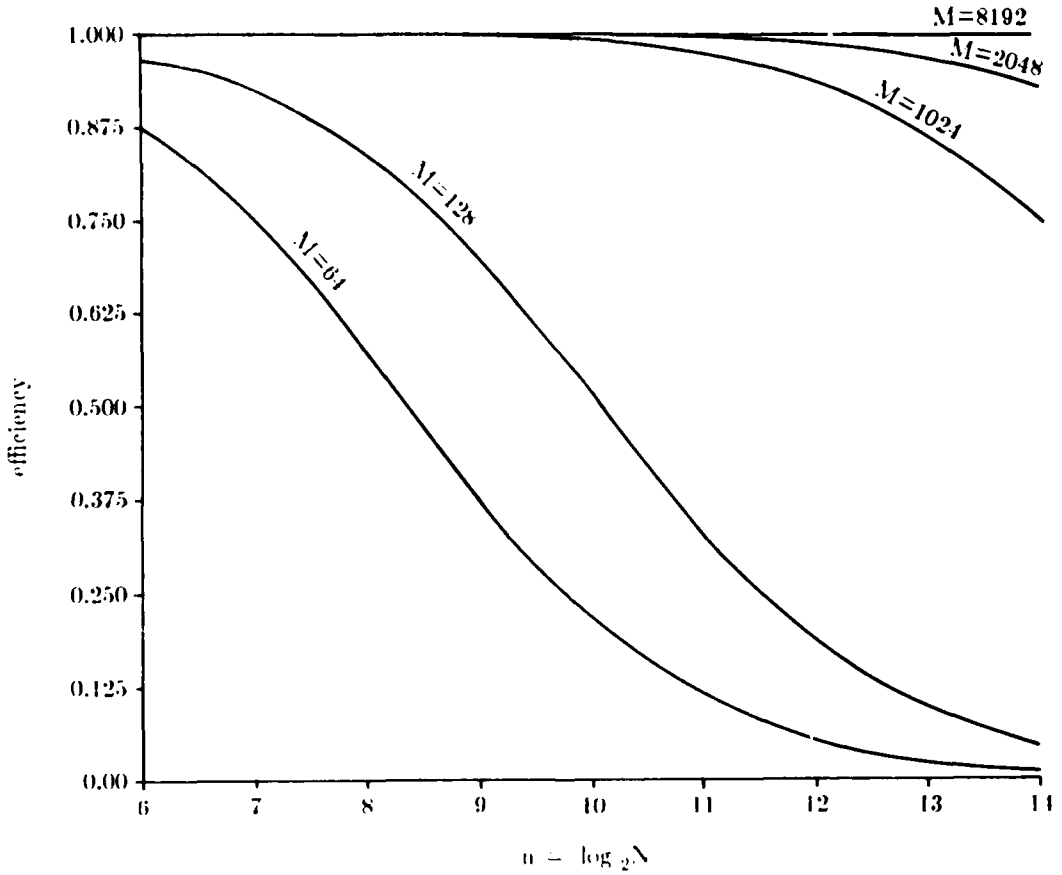


Fig. 3.3.5. Clustering Algorithm: Ellicicncy

of recursive doubling are used. At step i, $1 \leq i \leq \log_2 N$, the number of PEs performing the additions is $N/2^i$. Summing over the $\log_2 N$ steps gives N−1. In the stage of testing whether pchange is less than t or not, only PE 0 is enabled. For the rest of the computation operations, all N PEs are enabled. Figure 3.3.6 shows the utilization under the simplifying assumptions. In this example, because most computation steps use all N PEs,

$$\sum t_x P_x \simeq N * c_N(M^2) \quad .$$

Therefore, here $U_N(M^2) \simeq c_N(M^2)/T_N(M^2)$.

Efficiency is directly affected by both overhead and utilization. As one may expect, efficiency will decrease as overhead increases and/or utilization decreases. If, for a given set of parameters, efficiency is low, then the overhead and utilization may be examined to determine factors contributing to the low efficiency. For the clustering algorithm, both overhead and utilization cause efficiency to decrease as N increases.

(e) Price: The price for the clustering example is

$$P_N(M^2) = P_t * T_N(M^2) + P_i * [N * P_{PE} + N * P_s]$$

where P_t = cost of a unit of execution time

P_{PE} = cost of a PE

P_s = cost of a network switch

P_i = relates total implementation cost to hardware costs

Assuming a single stage cube network is used, the number of switches is N. Figure 3.3.7 shows the $\log_2$ of the price under the simplifying assumptions plus

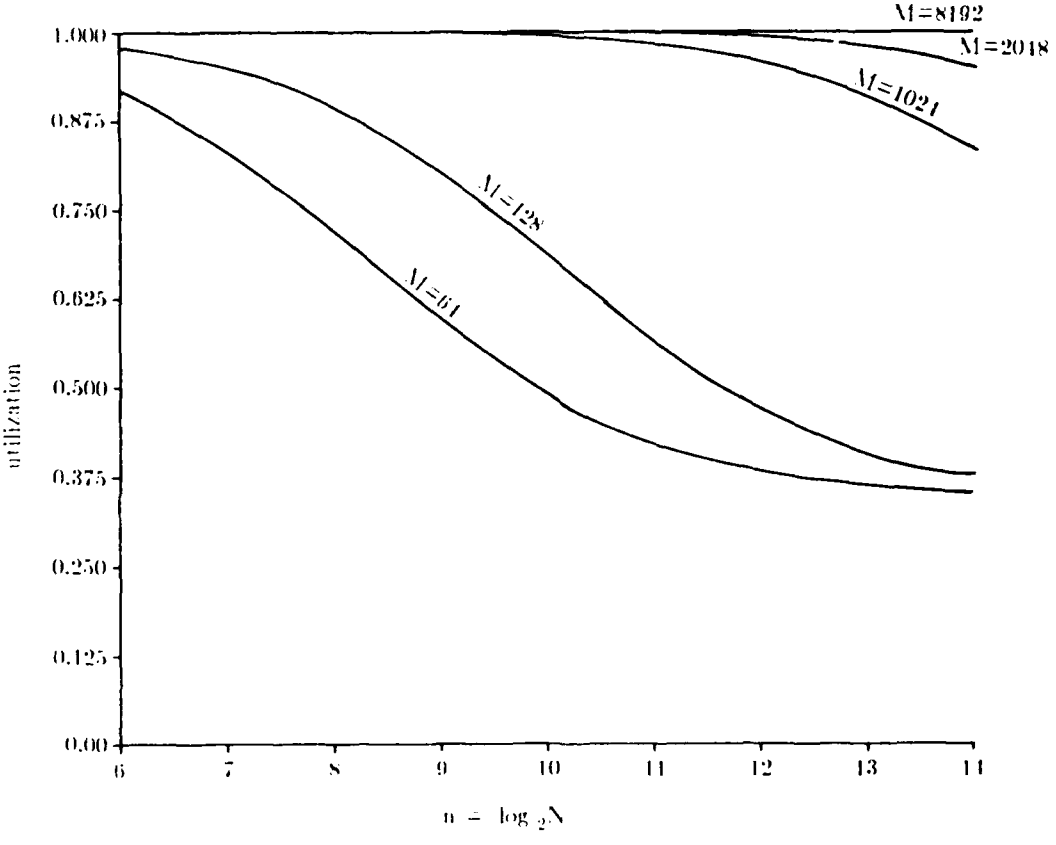


Fig. 3.3.6. Clustering Algorithm: Utilization

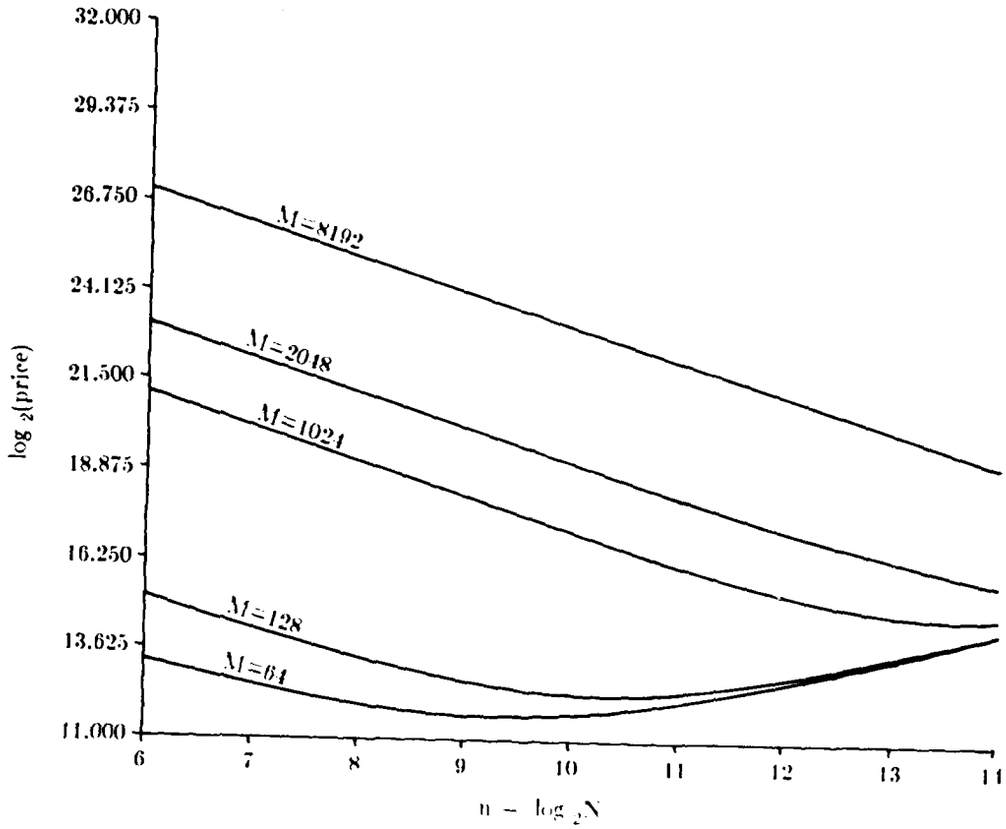


Fig. 3.3.7. Clustering Algorithm: Price

the assumptions that $P_t = P_i = 1$, and $P_{PE} = 32 * P_s = 1$. For small M, the curve has a minimum in the range $2^6 \leq N \leq 2^{11}$. Furthermore, the optimal N is greater for large images than for small images. This occurs because for large images, execution time continues to decrease significantly as N increases, while for smaller images, the rate at which $T_N(M^2)$ decreases falls off for large N.

The generalized price for the clustering algorithm, under the same assumptions, is given by

$$P'_N(M^2) = \frac{\alpha}{\alpha+1} * P_t * T_N(M^2) + \frac{1}{\alpha+1} * P_i * [N * P_{PE} + N * P_s]$$

where α expresses the relative importance of execution time versus system cost. Figure 3.3.8 shows the generalized price as a function of N and α for a 1024x1024 image. As expected, the optimal value of N shifts to the right as execution time becomes more critical than system cost and to the left when system cost dominates ($\alpha < 1$).

3.4 Parallel Architecture

From the discussion of the previous three sections, a multiprocessor system which can be dynamically reconfigured as one or more independent SIMD/MIMD submachines of different sizes is needed to implement the supervised relaxation algorithm. In this section, the partitioning of the interconnection networks, cube network and augmented data manipulator network, is considered. Typically, when the number of processing elements in the multiprocessor system increases, the data communications between the PEs become more and more important. Especially for multiple submachines of SIMD and MIMD modes, it is required that different submachines can simultaneously execute their instructions and do not interfere with each other.

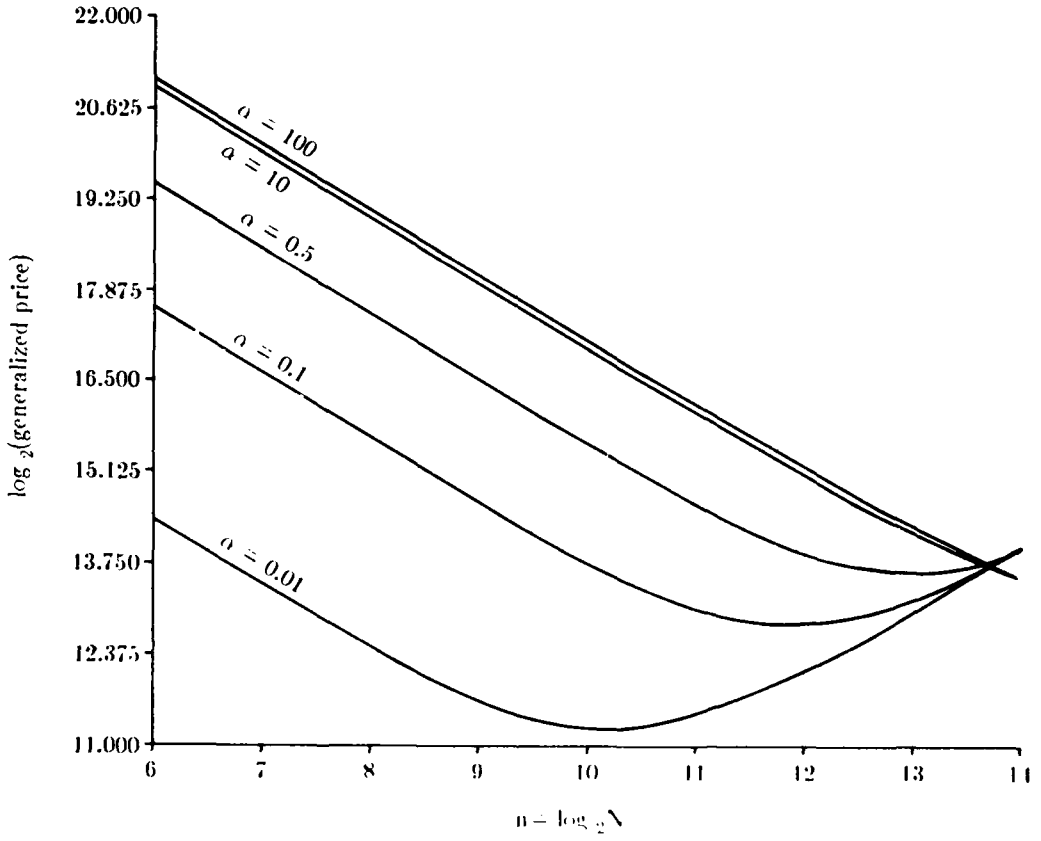


Fig. 3.3.8. Clustering Algorithm: Generalized Price

But, they still can cooperate with each other in the results through the interconnection network after certain stages of computations. The routing schemes presented below for both cube network and ADM network and hybrid type network are quoted from [16,17,18,19]. The network adopted is so well suited to the supervised relaxation algorithm that substantial speedup by the multiprocessor system is expected.

3.4.1 Hybrid Network

As shown in Fig. 3.4.1, the hybrid cube network is an integration of two networks, the unidirectional packet-switched cube network and the bi-directional circuit-switched cube network. This network is suitable for the parallelism modes configured as shown in Fig. 3.2.1 because large blocks of input data can be pipelined through the lower half network and fast data and instruction fetches can be provided by the upper half network. The upper half of the hybrid cube network is a PE-to-PE configuration in which the cube network is wrapped around and connected to N PEs. There is a local memory module associated with each PE. The lower half of the hybrid cube network is a PE-to-memory configuration in which N PEs are connected on one side and N memory modules are connected on the other side. The memory spaces in the N memory modules shared by the N PEs are much larger than the local memory modules in the N PEs. The interchange boxes at the mth stage ($m = \log_2 N$) can be set to connect to the upper half or the lower half of the hybrid cube network by setting the interchange boxes to straight or exchange; they each have one input port connected to a PEs.

The advantage of the PE-to-PE configuration is fast local memory accesses. For SIMD mode, data are stored in the local memory. Therefore,

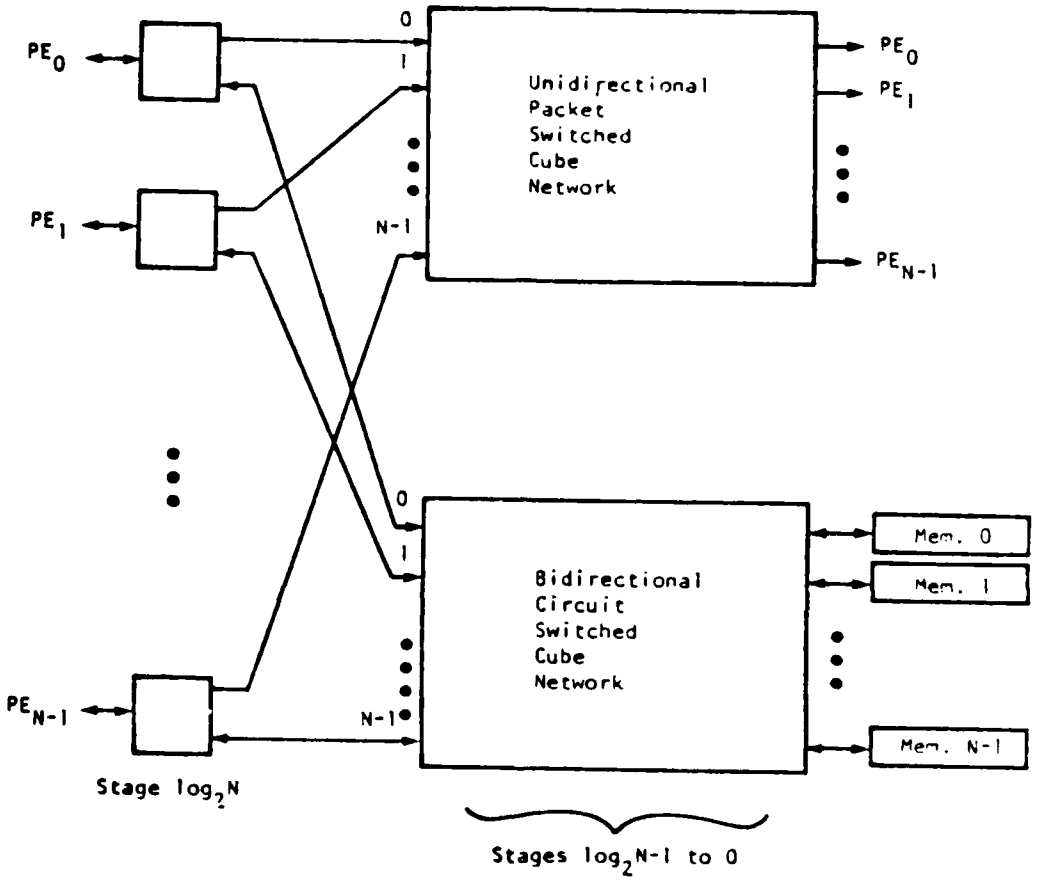


Fig. 3.4.1. Hybrid Cube Network [19]

this configuration can provide fast instruction and data fetches. For the PE-to-memory configuration, N PEs share large blocks of data stored in N memory modules and can vary the amount of memory used by each processing element.

For the packet-switched cube network, a packet makes its way from stage to stage, releasing links and interchange boxes immediately after using them. It is good for MIMD mode which needs a frequently changing path through the network when performing window-type operations. Therefore, every PE can request data from a neighboring PE by sending a message via the interconnection network and the requested PE can respond accordingly. As a result, it can reduce contention incurred by sending two packets to the same input port of the interchange box or dispatching two packets from the same output port of the interchange box. For the circuit-switched cube network, a complete circuit is established from input port to output port for one particular path. It is good for SIMD mode which can provide parallel data transfers via the interconnection network and also good for transferring large blocks of data from the shared memory modules to PEs. Therefore, data transfers can be pipelined through the network. Once this path is established, the only delay is propagation delay.

The upper half of the hybrid cube network is an unidirectional network. Since the inputs and outputs of the network are connected to PEs, the unidirectional network is sufficient. For the lower half of the hybrid cube network, since large blocks of data need to be transferred between PEs and memory modules, a bidirectional network is necessary to provide this facility.

Actually, the hybrid cube network contains N input ports, 2N output ports and two size N generalized cube networks. It has m+1 stages labeled from m to 0. Interchange boxes in stage m divide the network into two halves.

That is, the interchange boxes can be set to connect to either the upper half or lower half of the network.

If the cube network is replaced by the ADM network, then the hybrid cube network becomes the hybrid ADM network. The ADM network is more powerful than the cube network. For the multistage cube network, there is only one path between a particular network input and output. But for the multistage ADM network, there are multiple paths between a given network input and output. Thus, the hybrid ADM network can reroute the packets through another path if the current established path is broken or has busy switching elements in it. It can perform any permutation that a multistage cube network can perform. However, in addition to these advantages for the hybrid ADM network, there are also additional implementation costs and control complexity that the hybrid cube network does not have.

If the multiprocessor system with the hybrid network is used to implement the supervised relaxation algorithm, the data inputs to the subalgorithms in blocks A, B, and C can be stored in the memory modules first. The input data tend to be large. Therefore, the memory modules can provide large storage space for them and data can be retrieved on demand. After using them, results produced can be transferred back to memory modules in order to save memory spaces in PEs for other purposes.

3.4.2 Partitioning the Cube Network into 2 MIMDs and 1 SIMD

Partitioning and routing schemes for the cube network are described in the Appendix.

For simplicity, assume there are N = 16 PEs. We want to partition these PEs into 2 MIMD submachines and 1 SIMD submachine for the implementation of the supervised relaxation algorithm in this multiprocessor system. One MIMD has 8 PEs and the other MIMD has 2 PEs. The SIMD has 4 PEs plus one PE for the control unit. Thus a total of 15 PEs have been used. Now, the remaining problem is how to partition the network into 2 MIMDs and one SIMD with the control unit associated with it. If the network can be partitioned into these three independent groups, then the PEs connected to this network are automatically partitioned into three corresponding independent submachines.

For MIMD mode with size 8, the addresses of these 8 ports must differ in 3 bit positions. For example, if port addresses from 0 to 7 are assigned to the MIMD machine, their addresses will all agree in only the most significant bit position. By setting to straight the interchange boxes in stage 3 corresponding to the input port addresses ranging from 0 to 7, one MIMD with size 8 is formed. Similarly, for MIMD mode with size 2, the addresses of these two must differ in 1 bit position. Let us say that port addresses, 10 and 11, form one MIMD. Therefore, two addresses will agree in the upper 3 bit positions. By setting to straight the interchange boxes in stages 3, 2, and 1 corresponding to these two port addresses, one independent MIMD with size 2 is formed. For SIMD, if port address 8 is selected as the control unit and port addresses ranging from 12 to 15 are selected for the processing PEs, then these latter 4 addresses agree in the most significant bit position. By setting to straight the interchange boxes in stage 3 corresponding to port addresses from 12 to 15, these 4 addresses form one independent group. PE 8 can broadcast instructions to these 4 by calculating the routing tag as follows:

$$R = S \oplus D_i = 8 \oplus 12 = 1000 \oplus 1100 = 0100$$

and

$$B = D_i \oplus D_j = 12 \oplus 15 = 1100 \oplus 1111 = 0011$$

(the notation is defined in the Appendix). For PE 8 by setting the interchange boxes in stage 3, 2, 1, and 0 to straight, exchange, broadcast, and broadcast, it can broadcast instructions to PEs 12, 13, 14, and 15. The result is shown in Fig. 3.4.2.

3.4.3 Partitioning the ADM Network into 2 MIMDs and 1 SIMD

Since the determination of routing tags can be found in [16,17], this section only overviews the advantages of the ADM network. Most important is the description of the partitioning of the ADM network into 2 MIMDs and 1 SIMD.

The advantages of the ADM network as described before are the multiple paths existing between a network input and output, and the additional permutations, as compared to the cube network, which the ADM network can perform.

The ADM network consists of N input ports, N output ports, and $m = \log_2 N$ stages with N switching elements per stage. Stages are numbered from $m-1$ to 0. Each switching element in stage i performs a straight connection and the PM2I (plus-minus 2^i) interconnection function which is defined as

$$PM_{+i}(j) = (j + 2^i) \bmod N$$
$$PM_{-i}(j) = (j - 2^i) \bmod N$$
$$\text{for } 0 \le j < N \quad \text{and} \quad 0 \le i < m$$

Therefore, each node j at stage i of the network has three output lines. But,

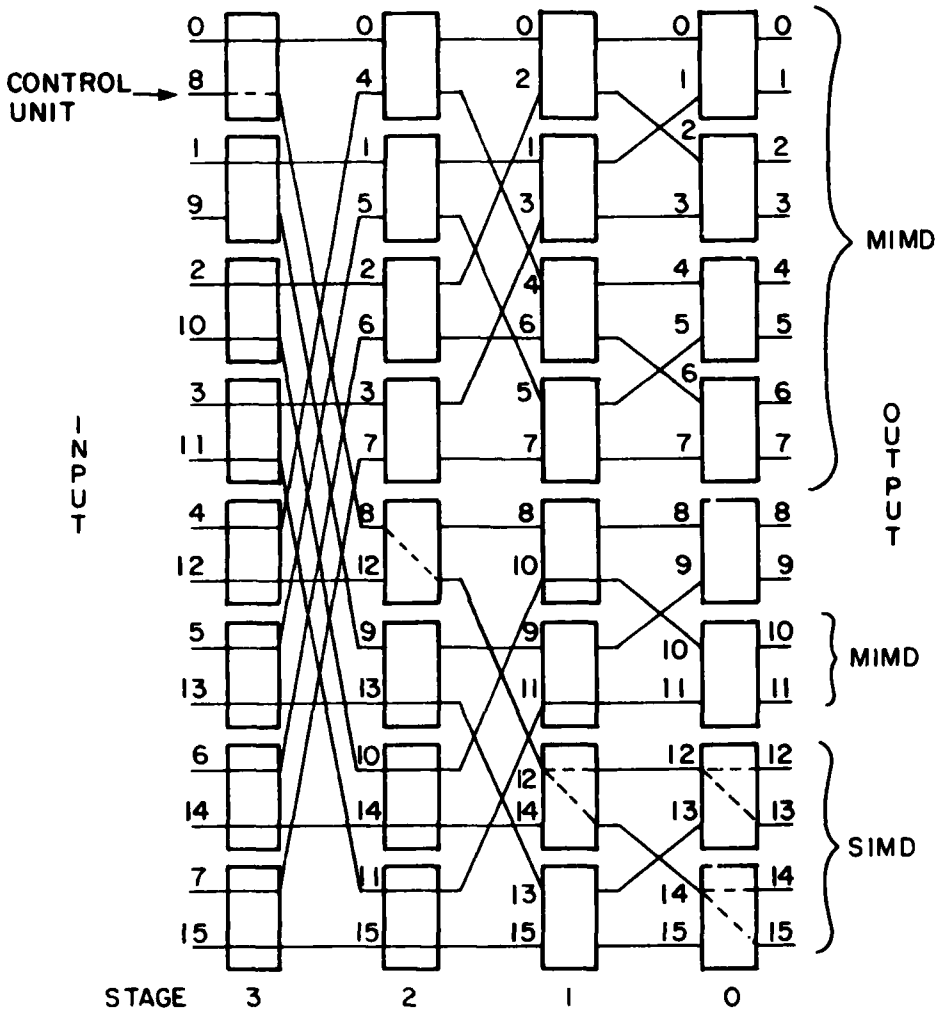


Fig. 3.1.2 Cube Network with 2 MIMDs and 1 SIMD

there are only two distinct data paths from each node in stage m-1 [17]. The ADM network is shown in Fig. 3.1.3. The individual switching elements in every stage can be controlled independently and routing tags are employed to distribute control of the network among the processors to avoid the bottleneck created by the centralized control unit.

Assume there are N = 16 PEs connected to the ADM network. The network can be partitioned into one SIMD machine with size 2, one MIMD machine with size 4, and one MIMD machine with size 2 as shown in Fig. 3.4.1. For the MIMD machine of size 4, if the port addresses include 1, 5, 9, and 13, set the corresponding switching elements in stages 0 and 1 to straight. For MIMD machine of size 2, if this group includes ports 7 and 15, set the corresponding switching elements in stages 0, 1, and 2 to straight. The principle of partitioning is that the size of each subnetwork must be a power of 2 and the addresses of the input ports in the subnetwork of size 2^s must agree in their lower order m−s bit positions. For SIMD mode, port 11 acts as the control unit to broadcast instructions to ports 2 and 4. The routing tag contains two parts: {R,B} = {11001, 10001}. In [16,17], calculation of the routing tags is described. Therefore, PE 11 sets the switching elements in successive stages to be -2^3, straight, straight, 2^1−type broadcast. Thus, it can broadcast instructions to PEs 2 and 4. The 2 MIMD modes and 1 SIMD mode in the independent subnetworks shown in Fig. 3.4.1 have the complete interconnection capabilities of the ADM network.

The important ability of this network is that it can dynamically reroute the message through alternate available paths to gain fault tolerance as well as improved throughput [17].

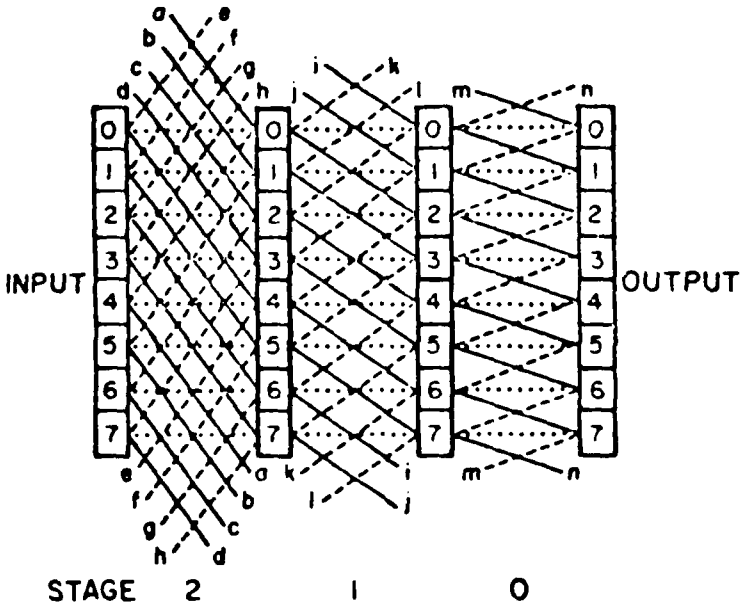
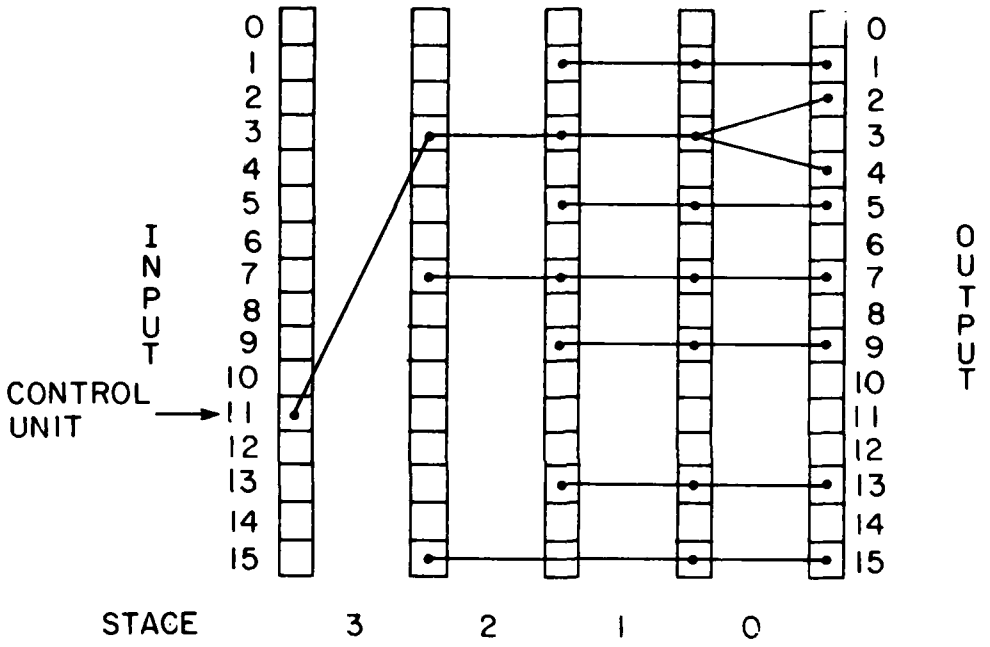


Fig. 3.4.3. ADM Network [16,17]



MIMD = {1,5,9,13}

MIMD = {7,15}

SIMD = {2,4}

Fig. 3.4.4. ADM Network with 2 MIMDs and 1 SIMD

3.5 Summary and Conclusions

In Section 3.1, it was shown that the maximum parallelism achieved at this level of modeling is reduced by the nature of the algorithm and especially by the scalar processes involved in the testing of the loop despite the availability of 1024 PEs. It also was shown that the average parallelism $\bar{h}$, which accounts for both concurrency of processing elements and for the overlap of array and scalar or control-flow instructions, is usually larger than the average vector parallelism $\bar{\bar{h}}$ which only accounts for the concurrency of processing elements and doesn't consider the overlap of array and scalar or control-flow instructions. If the number of processing elements is relatively small, the overlap of the control unit with the array processors may be significant. Otherwise, the overlap of the control unit with the array processors contributes very little to the degree of parallelism.

An interesting result of the analyses in this section was that the total time required to classify an image using a pipeline is greater than that using an SIMD architecture even though the utilization of the PEs is greater for the pipeline. The reason is that the overhead due to implementing the parallelism for the pipeline is greater. The overhead includes scalar operations executed at each stage of the pipeline and data transfers between stages of the pipeline. In SIMD mode, PE address and general masking operations cost nothing in the total algorithm run time if we assume that masking operations are decoded in the control unit. This property is also modeled with S-Nets. Therefore, the transition firing time is not increased to accommodate mask markings that are not all 1's. Data transfer time and data transfers through the interconnection network can also be modeled with S-Nets, such as the data transfers in vector sum example. Based on the S-Net model, system throughput can be increased

by maximizing the $\bar{h}$ and $\bar{\bar{h}}$ measures.

Also in this section, we modeled a complex algorithm (maximum likelihood classification) with S-Nets on SIMD and pipeline architectures. Some quantitative measures such as parallelism and execution time were considered in conjunction with S-Nets. Due to the availability of quantitative measures, we can make direct comparisons between two architectures based on a given image size and number of PEs. In general, for image processing operations which are not window-type operations, the higher the dimensionality of the remote sensing data and the more classes represented in the image, the greater the potential benefits to be derived from SIMD implementation of the process.

Section 3.2 discussed the comparison between execution times of SIMD and MIMD processors based on the analysis of explicit and implicit operations embedded in an algorithm. In Section 3.2.4 it was demonstrated that the MIMD mode is better than SIMD mode for the algorithms which are not suitable for lockstep operation. Of course, MIMD mode needs a more complicated control strategy and has more complicated methods for data transfers than SIMD mode. In SIMD mode, every step is synchronized and the control strategy is simpler. In general, algorithms which have window-type operations and do not have operators whose execution times are argument-dependent are suitable for SIMD mode because they can take advantage of overlapping of scalar and array operations, parallel data transfers, synchronized operation, and simpler control schemes without loosing the better utilization of PE resources. Algorithms which do not have window-type operations and have operators whose execution times are argument-dependent are suitable for MIMD mode because they can avoid the interrupt operations for asynchronous data transfers and deadlock prevention schemes, achieve better utilization of

PE processing power, and minimize execution times. For algorithms which have window-type operations and are suitable for MIMD mode, data transfers can be executed in parallel before performing any data computation or can be executed in a more general fashion though interrupt operations provided that the complicated control scheme of the system is available. We assume that asynchronous data transfers through interrupt operations and control schemes can be supported by the hardware/software.

The supervised relaxation algorithm contains several subalgorithms, some of which have execution times that are argument-dependent. These subalgorithms are suitable for MIMD mode in order to most efficiently utilize the processing power of PEs and minimize execution times. On the other hand, subalgorithms with execution times which are not argument-dependent are most suitable for SIMD mode in order to take advantage of the simpler control strategy and lockstep operations. PASM [26,27], a partitionable, reconfigurable SIMD/MIMD multimicroprocessor system, offers an interesting environment in which to implement the supervised relaxation algorithm. It can be configured as one SIMD submachine and two MIMD submachines of different sizes for the earlier processing stages and later can be reconfigured as an SIMD machine with maximum PEs in order to avoid a bottleneck due to a particularly complex operation. From Table 3.2.3, it was seen that the speedup of both SIMD and MIMD submachines will be close to $(M/n)^2$, which is the number of PEs, if the subimage size is large enough so that the overhead caused by parallelism is not a dominant factor.

In Section 3.3, analysis of a representative algorithm (clustering) emphasized system performance as a function of problem size and system size. Although the system performances are for SIMD mode, it can be generalized to

MIMD mode as long as data transfers are executed in parallel before performing data computation, and the computation time for MIMD mode must take into account scalar operations such as loop counting, branching, and subroutine calls and returns. The other processing environment of MIMD mode is the same as that described in Section 3.2. We recall that the subalgorithms in blocks A, B, and C as shown in Fig. 3.2.1 can be executed independently and simultaneously because the input data and results of these three subalgorithms are independent of each other. For systems aimed at real-time processing, it is best that these subalgorithms produce their results almost at the same time to provide input data for block D. Based on the "execution time" measure, we can assign different numbers of processing elements for subalgorithms in blocks A, B, and C to synchronize their outputs and minimize the execution time. But by considering execution time alone, it is possible to select the number of PEs such that the marginal improvement in performance is very small or, conversely, such that significant improvements may be sacrificed. The execution time does not directly address issues related to how effectively the system resources are being used. In general the "speed-up", "efficiency", and "utilization" measures used together can achieve better utilization of PE resources and synchronize the outputs. By considering the total number of PEs assigned to blocks A, B, and C as system cost and the required execution time to synchronize the outputs as execution-time cost, the "price" and "generalized price" measures can provide a means to choose the number of PEs which seems to be a compromise between execution time and system cost.

In Section 3.4, we considered two interconnection networks which can provide the communication among the PEs in the multiprocessor system and

described how these two networks can be reconfigured as 2 MIMDs and 1 SIMD subnetworks to support the system to implement the supervised relaxation algorithm.

CHAPTER 4 - CONCLUSIONS AND SUGGESTIONS

The supervised relaxation operator was demonstrated successfully as a mechanism for combining the information from multiple ancillary data sources with the information from spectral data and spatial context in classifying of multispectral imagery. In Chapter 2, the method was described and the convergence of the iterative algorithm was established. The final labeling results from convergence of evidence, reaching consistent labeling. The key inputs to the supervised relaxation operator are a quantitative characterization of initial probabilities computed from the spectral data and conditional probabilities computed from the contextual information. The initial probabilities were obtained in the penultimate step of the maximum likelihood classification using spectral data; the conditional probabilities were calculated from another source of data known to be correct. The experimental results showed that the performance of the method using spatial contextual information was only slightly better than that obtained from the maximum likelihood classifier; the performance with one ancillary information source was much better than previously obtained; and the performance measure with two ancillary information sources was still better. These results demonstrate that the supervised relaxation algorithm is a useful technique to integrate information from the various sources and achieve a ground cover classification

which is both accurate and consistent in the face of inconsistencies which may exist among the data components.

In Chapter 3, S-Nets were used to describe the maximum likelihood algorithm implemented in SIMD and pipeline modes of parallelism. Analysis of the S-Net models showed that the overhead incurred by the pipeline causes longer execution time than SIMD mode. PE address and general mask operations, which for SIMD are decoded in the control unit and cost nothing in the algorithm run time, can also be modeled with S-Nets, as can the data transfer time and data transfers through the network.

Some quantitative measures such as average parallelism and execution time were also developed and used to make direct comparisons between two architectures. Detailed descriptions and analyses of implicit operations, explicit operations, loop counting operations, parallel transfers, and interrupt operations occurring in SIMD and MIMD modes of parallelism were presented in Section 3.2. Based on these analyses, the comparison of execution times derived can lead to the right decision concerning which mode of parallelism, SIMD or MIMD, is best suited to one specific algorithm. In general, algorithms which have window-type operations and do not have operators whose execution time are argument-dependent are suitable for SIMD mode because they can make use of overlap of scalar and array operations, parallel data transfers, synchronized operation, and simpler control scheme. Algorithms which do not have window-type operations and have operators whose execution times are argument-dependent are best suited to MIMD mode. Even for algorithms which have window type operations and have operators whose execution times are argument-dependent, MIMD mode is still suitable as long as data transfers can be executed in parallel fashion before performing data computation.

Section 3.3 described the determination of the optimal number of processing elements for a given image size based on measures of evaluating the performance of algorithms for SIMD machines. An important concept is that the number of PEs can be chosen subject to the relative importance of system cost and execution time.

Finally, two multistage interconnection networks were described, the cube network and ADM network, which provide the communication medium for the multiprocessor system and can be configured to operate as subnetworks supporting complex tasks.

The neighborhood contributions from the four nearest neighbors have been used in the current study. For further research, parallel implementations may be developed to utilize neighborhood relations beyond just near neighboring pixels to combine the contextual information in the algorithm. Equal weighting constants, d_{ij}, have been assigned to the four nearest neighbors; i.e., these four neighboring pixels have equal degree of influence in the neighborhood contribution. If the weighting constants can be dynamically adjusted to allow different neighbors to have different degrees of influence on the current pixel classification, the classification accuracy expected may be better. Furthermore, in a more complicated case in which one ancillary data source is felt to be more accurate than another, it is possible to assign two different weighting constants to the ancillary variables in the supervised relaxation process.

The current modeling based on SIMD and pipeline architectures appears to work quite well. A future challenge is to use S-Nets to model control strategies or operating systems for various type architectures. Also, further investigations should exploit algorithms on MIMD architectures and on MSIMD architectures

in which SIMDs operates asynchronously. If the total elapsed time of the S-Net the transition sequences and the average parallelism of the S-Net can be quantitatively determined, together with the description presented in Section 3.2, the results may provide a more reliable way to decide which architecture model is best suited to a particular algorithm.

LIST OF REFERENCES

LIST OF REFERENCES

[1] R.M. Hoffer and staff, "Computer-aided analysis of SKYLAB multispectral scanner data in mountainous terrain for land use, forestry, water resource, and geologic applications," Tech. Rept. No. 121275, LARS, Purdue University, 1975.

[2] J. Richards, D. Landgrebe, and P. Swain, "A means for utilizing ancillary information in multispectral classification," *Remote Sensing of Environment,* Vol. 12, pp. 463-477. 1982.

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

[4] B. Schachter, A. Lev, S. Zucker, and A. Rosenfeld, "An application of relaxation methods to edge reinforcement," *IEEE Trans. Syst., Man, Cybern.,* Vol. SMC-7, No. 11, pp. 813-816, Nov. 1977.

[5] A. Lev, S. Zucker, and A. Rosenfeld, "Iterative enhancement of noisy images," *IEEE Trans. Syst., Man, and Cybern.,* Vol. SMC-7, No. 6, pp. 435-442, June 1977.

[6] A. Rosenfeld, "Iterative methods in image analysis," *Proc. 1978 IEEE Conf. Pattern Recognition and Image Processing,* pp. 181-187, Jan. 1978.

[7] S. Zucker and J. Mohammed, "Analysis of probabilities relaxation labeling processes," *Proc. 1978 IEEE Conf. Pattern Recognition and Image Processing,* pp. 307-312, May 1978.

[8] S. Zucker, E. Krishnamurthy, and R. Haas, "Relaxation processes for scene labeling: convergence, speed, and stability," *IEEE Trans. Syst., Man, Cybern.,* Vol. SMC-8, No. 1, pp. 41-48, Jan. 1978.

[9] R. Hummel an S. Zucker, "On the foundations of relaxation labeling processes," *Proc. 1980 IEEE Conf. Pattern Recognition,* pp. 50-53, March 1980.

[10] S. Peleg and A. Rosenfeld, "Determining compatibility coefficients for curve enhancement relaxation processes," *IEEE Trans. Syst., Man, Cybern.*, Vol. SMC-8, No. 7, pp. 548-555, July 1978.

[11] H. Yamamoto, "A method of deriving compatibility coefficients for relaxation operators," *Computer Graphics and Image Processing*, Vol. 10, pp. 256-271, 1979.

[12] J. Richards, D. Landgrebe, and P. Swain, "Pixel labeling by supervised probabilistic relaxation," *IEEE Trans. Pattern Analy. and Machine Intel.*, Vol. PAMI-3, No. 2, pp. 188-191, March 1981.

[13] R.A. Hummel and S.W. Zucker, "On the foundations of relaxation labeling processes," *IEEE Trans. PAMI*, Vol. PAMI-5, No. 3, pp. 267-287, May 1983.

[14] A.J. Krygiel, "Synchronous nets for single instruction stream - multiple data stream computers," D.Sc. dissertation, Sever Institute of Technology, Washington University, St. Louis, MO, May 1980.

[15] A. J. Krygiel, "Synchronous nets for single instruction stream - multiple data stream computers," *Proc. 1981 Int'l. Conf. Parallel Processing*, Aug. 1981, pp. 266-273.

[16] R.J. McMillen, H.J. Siegel, "Routing schemes for the augmented data manipulator network in an MIMD system," *IEEE Trans. Compt.*, Vol. C-31, pp. 1202-1214, Dec. 1982.

[17] H.J. Siegel and R.J. McMillen, "Using the augmented data manipulator network in PASM," *Computer*, Vol. 14, No. 2, pp. 25-33, Feb. 1981.

[18] H.J. Siegel and R.J. McMillen, "The multistage cube: A versatile interconnection network," *Computer*, Vol. 14, No. 12, pp. 65-76, Dec. 1981.

[19] R.J. McMillen and H.J. Siegel, "The hybrid cube network," *IEEE Proc. of the Distri. Data Acquis., Compu., and Control Sym.*, pp. 11-22, Dec. 1980.

[20] J. Richards, D. Landgrebe, and P. Swain, "On the accuracy of pixel relaxation labeling," *IEEE Trans. Syst., Man, and Cybern.*, Vol. SMC-11, No. 4, pp. 303-309, April 1981.

[21] P.H. Swain, "Fundamentals of pattern recognition in remote sensing," in *Remote Sensing: The Quantitative Approach*, P. H. Swain and S. M. Davis, Eds., McGraw-Hill, New York, 1978.

[22] L.J. Siegel, H.J. Siegel, P.H. Swain, G.B. Adams III. A.E. Feather, G.M. Lin, and M.R. Warpenburg, "Parallel Image Processing/Feature Extraction Algorithms and Architecture Emulation: Interim Report for Fiscal 1981," TR-EE 81-35, Purdue Univ., School of Electrical Engineering, Oct. 1981.

[23] J.T. Kuehn, "A more accurate complexity analysis technique," informal communications, Sept. 1982.

[24] L.J. Siegel, H.J. Siegel, and A.E. Feather, "Parallel processing approaches to image correlation," *IEEE Trans. Comput.*, Vol. C-31, pp. 208-218, Mar. 1982.

[25] P.T. Mueller, Jr., L.J. Siegel, and H.J. Siegel, "Parallel algorithms for the two-dimensional FFT," *Proc. 5th Int'l. Conf. Pattern Recognition*, pp. 497-502, Dec. 1980.

[26] D.L. Tuomenksa, G.B. Adams III, H.J. Siegel, and O.R. Mitchell, "A parallel algorithm for contour extraction: advantages and architectural implications," *IEEE Comput. Soc. Conf. Computer Vision and Pattern Recognition*, pp. 336-344, June 1983.

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

[28] R.O. Duda and P.E. Hart, *Pattern Classification and Scene Analysis*, Wiley, New York, 1973.

[29] L.J. Siegel, H.J. Siegel, an P.H. Swain, "Performance measures for evaluating algorithms for SIMD machines," *IEEE Trans. Software Engineering*, Vol. SE-8, pp. 319-331, July 1982.

[30] K.E. Batcher, "MPP - a massively parallel processor," *Proc. 1979 Int'l Conf. Parallel Processing*, Aug. 1979, p. 249.

APPENDIX

APPENDIX -
PARTITIONING AND ROUTING SCHEMES OF THE CUBE NETWORK [18]

The generalized cube network has N inputs, N outputs, and $m = \log_2 N$ stages. Each interchange box can be set to one of the four legitimate configurations shown in Fig. A.1. The m cube interchange functions are defined as

$$\text{cube}_i(P_{m-1} \cdots P_1 P_0) = P_{m-1} \cdots P_{i+1} \bar{P}_i P_{i-1} \cdots P_1 P_0$$

where

$$0 \leq i < m$$

$\bar{P}_i$ means the complement of bit P_i. Stage i of the generalized cube network contains the cube_i interconnection function, i.e., i/o lines of each box differ in the ith bit position as shown in Fig. A.1.

The cube network can be partitioned into independent groups. The PEs in a group must agree in m-s bits if this group has 2^s PEs and $m = \log_2 N$. For example, if the cube network has $N = 8$ inputs and outputs, it may be partitioned into two groups: group A consists of ports 0 to 3 and group B consists of ports 4 to 7. By setting all of the interchange boxes in stage 2 to straight, these two groups are isolated as shown in Fig. A.2. If the interchange boxes in the other stage are set to straight, then two independent groups other than the one in Fig. A.2 are formed as shown in Fig. A.3 and in Fig. A.4.

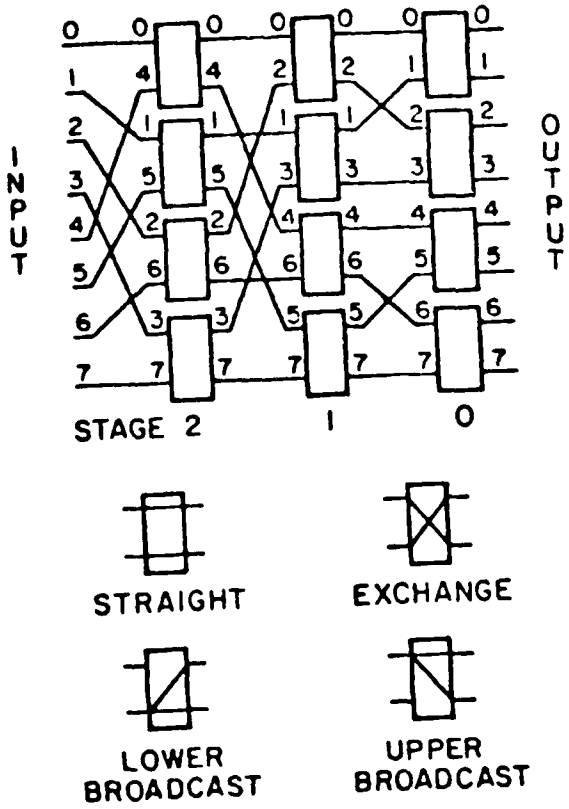


Fig. A.1. Generalized Cube Network and 4 Functions of Interchange box

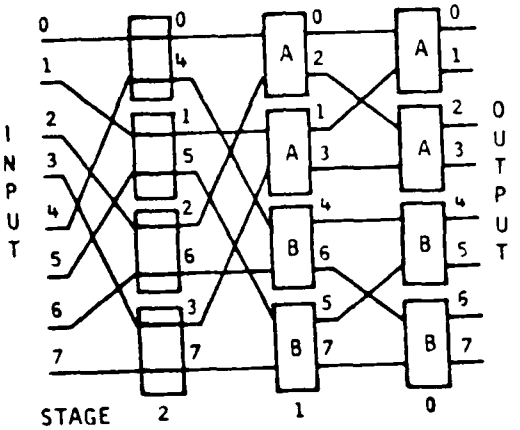


Fig. A.2. Set Boxes in Stage 2 to Straight [18]

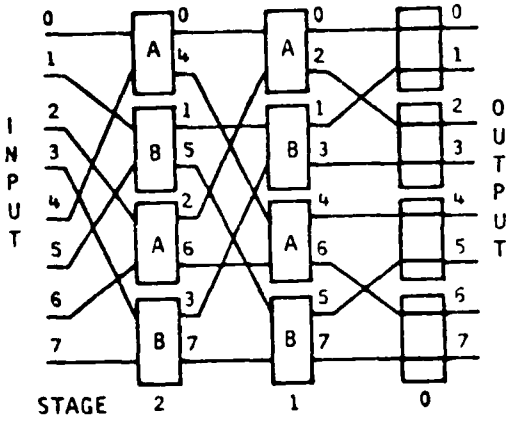


Fig. A.3. Set Boxes in Stage 0 to Straight [18]

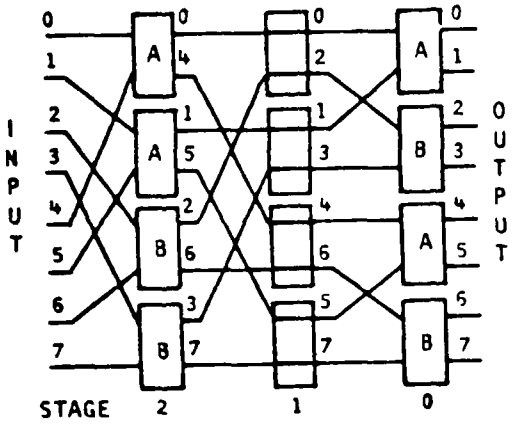


Fig. A.4. Set Boxes in Stage 1 to Straight [18]

Each subnetwork has the properties of a cube network and can be further subdivided to form smaller independent subnetworks. As shown in Fig. A.5, the subnetwork B in Fig. A.2 is further divided to form two independent subnetworks C and D.

Any type of a centralized control unit would create a bottleneck, but if control is distributed among the processors, each is responsible for determining its own control signals. Therefore, the purpose of using routing tags as headers on messages is to allow network control to be distributed among the processors. If the processors using the network are configured as an SIMD machine, their nonbroadcasting communication needs take the form of interconnection functions. An interconnection function is a specification of the way all of the N input ports are connected to the N output ports. An interconnection function is said to be admissible by the network if there are no conflicting requests for interchange box settings. In establishing an admissible interconnection function, routing tags are used by all active processors simultaneously. In MIMD mode, the routing requests occur at random intervals.

For one-to-one connection, the routing tag, T, is calculated as $T = S \oplus D$ in which S is the input port and D is the output port. The operator $\oplus$ denotes "exclusive or." For example, if $S = 5 = 101$ and $D = 3 = 011$, then $T = S \oplus D = 101 \oplus 011 = 110$. The interchange box in stage i only checks T_i which is the ith bit of routing tag T. If $T_i = 1$, set interchange box to exchange. If $T_i = 0$, set interchange box to straight. Fig. A.6 shows the path established between input port 5 and output port 3. The incoming tag is also the same as the return tag. Therefore, it can implement handshaking.

For one-to-many broadcast, the routing tag contains two parts: R contains routing information and B contains broadcast information. R contains m bits,

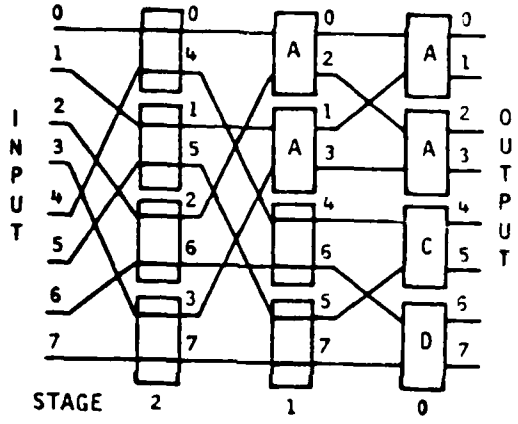


Fig. A.5. Subnetwork B is Divided to C and D [18]

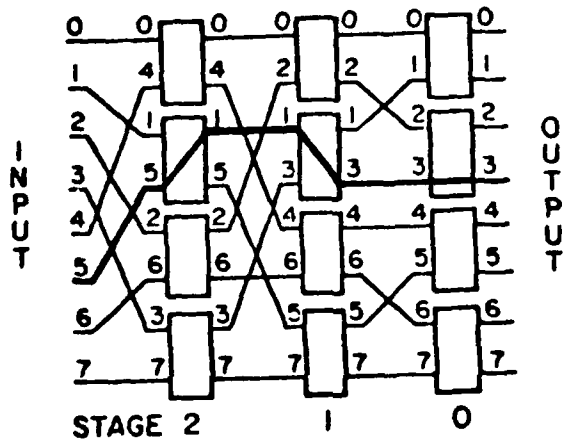


Fig. A.6. Connect Input Port 5 to Output Port 3

(T = 101 ⊕ 011 = 110)

so does B. For broadcasting, the destination group must be a group of size equal to 2^j. In this group, there must be at most j bits that disagree among any pair of the addresses, and m-j bit positions in which all these addresses must agree. For example, input port $S = 5 = 101$ broadcasts messages to output ports 2, 3, 6, and 7, in which all addresses agree in the first bit position (the least significant bit is the 0th bit). Then, $R = S \oplus D_i = S \oplus D_1 = 101 \oplus 010 = 111$ where D_i is any one of 4 destination addresses and $B = D_i \oplus D_k = 010 \oplus 111 = 101$ where D_i and D_k must have hamming distance of 2. The interchange boxes in stage i check B_i first. If $B_i = 1$, set interchange box to either upper broadcast or lower broadcast. If $B_i = 0$, then check R_i. If $R_i = 0$, set interchange box to straight, otherwise set exchange. Fig. A.7 shows input port, $S = 5$, broadcasts messages to output ports, $D = 2, 3, 6$ and 7.

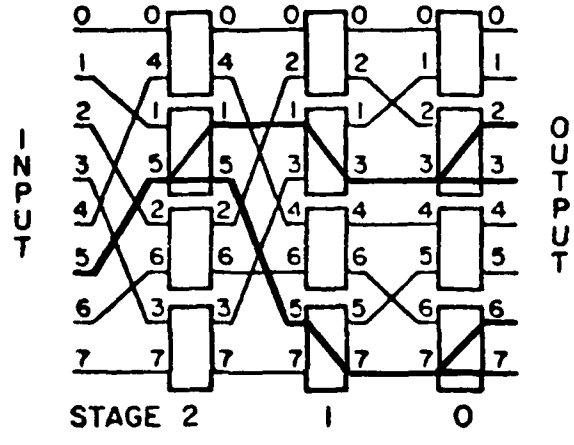


Fig. A.7. S = 5 Broadcasts Message to D = 2,3,6, and 7

END

FILMED

6-86

DTIC