Contextual Classification of Multispectral Remote Sensing Data Using a Multiprocessor System

PHILIP H. SWAIN, MEMBER, IEEE, HOWARD JAY SIEGEL, MEMBER, IEEE, AND BRADLEY W. SMITH, STUDENT MEMBER, IEEE

Abstract—A statistical model of spatial context is described and procedures for classifying remote sensing data using a context classifier are outlined. Experimental results are presented. Because the computational requirements of the context classifier are very large, its implementation on multiprocessor systems is investigated. Some of the special considerations necessary for such implementations are described, with particular reference to implementation on an array of Control Data Corporation Flexible Processors.

I. Introduction

FOR MORE THAN a decade, efforts to extract information from multispectral remote sensing image data have proved increasingly successful. To a large extent, these efforts have focused on the application of pattern recognition techniques to the multispectral measurements made on individual ground resolution elements, i.e., scenes have been classified pixel-by-pixel based on the measurement vectors associated with the individual pixels [1]. Progress has been achieved through development of increasingly sophisticated methods for extracting information from the spectral domain to characterize the classes of interest.

However, there are many applications for which the classes of interest can be better characterized if the spatial information in the remote sensing data is utilized in addition to the spectral information. Characteristic spatial features include, for example, shape, texture, and structural relationships. Some interesting and useful research has been accomplished in recent years in the direction of incorporating spatial information into the data analysis process [2]-[4].

One way to approach spatial information in image data is to recognize that the ground cover associated with a given pixel, i.e., its "class," is not independent of the classes of its neighboring pixels. Stated in terms of a statistical classification framework, there may be a better chance of correctly classifying a given pixel if, in addition to the spectral measurements associated with the pixel itself, the measurements and/or classifications of its "neighbors" are considered as well. Notice that at some point "neighborhood" must be explicitly defined.

If the objects in the scene tend to be rather large relative to the resolution of the sensor, i.e., each object is likely to consist of many spectrally similar pixels, this fact can be exploited nicely by applying a combination of scene segmentation techniques and sample classification (sometimes called "per-field" classification) [3]. More generally, the image can be considered a two-dimensional random process and the characteristics of this process incorporated into the classification strategy. This is the objective of the approach described here, in which a form of compound decision theory is employed to improve scene classification through use of a statistical characterization of context. This work is an extension of an idea by Welch and Salter [5].

As increasingly complex forms of data and data analysis methods are employed, the computational requirements tend to become more demanding. Although improvement in the raw speed of digital computer components can be exploited to some extent to meet these requirements, it is clear that evolving computer architectures, especially those involving multiple processing elements, have much to offer. The context classifier described here has computational requirements which are severe and rapidly become more so as the size of the contextual neighborhood is expanded. It is a natural candidate, therefore, for multiprocessor implementation.

II. The Context Classifier

The image data to be classified is assumed to be a two-dimensional $N_1 \times N_2$ array of multivariate pixels. Associated with the pixel at "row i" and "column j" is the multivariate measurement vector $X_{ij} \in R^n$ and the true state or class of the pixel $\theta_{ij} \in \Omega = \{\omega_1, \cdots, \omega_m\}$. The measurements have class-conditional densities $f(X \mid \omega_i)$, $i = 1, 2, \cdots, m$, and are assumed to be class conditionally independent. The objective is to classify the $N = N_1 \times N_2$ observations in the array.

The action (classification) determined by the classifier for pixel (i, j) is denoted by $a_{ij} \in \Omega$. To pursue a Bayesian (minimum risk) strategy, let the loss incurred by taking action a_{ij} when the true class is θ_{ij} be denoted by $L(\theta_{ij}, a_{ij})$ for some fixed nonnegative function $L(\cdot, \cdot)$. The loss associated with classification of the array is defined to be the average loss incurred over the N classifications in the array:

$$\frac{1}{N} \sum_{i,j} L(\theta_{ij}, a_{ij}). \tag{1}$$

In the most general case, the action a_{ij} may depend on all of the observations in the array. Let $\underline{X}$ denote this "vector of vectors," i.e., a vector the components of which are measure-

Manuscript received January 3, 1979; revised December 13, 1979. This work was supported in part by the National Aeronautics and Space Administration under Contract NAS9-15466.

The authors are with the School of Electrical Engineering and Laboratory for Applications of Remote Sensing, Purdue University, West Lafayette, IN 47907.

0196-2892/80/0400-0197$00.75

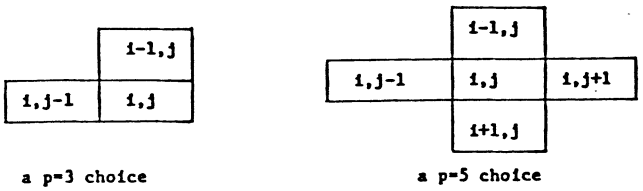

Fig. 1. A p-pixel neighborhood.

ment vectors. Then the expected loss is

$$R(\underline{X}) = E\left[\frac{1}{N}\sum_{i,j} L(\theta_{ij}, a_{ij}(\underline{X}))\right]$$

$$= \frac{1}{N}\sum_{i,j} E\left[L(\theta_{ij}, a_{ij}(\underline{X}))\right]. \tag{2}$$

The goal is to find a decision rule (the rule for choosing a_{ij} based on $\underline{X}$) which minimizes $R(\underline{X})$.

When context is ignored, the action (classification) depends only on the measurement vector X_{ij} of the pixel to be classified, in which case $a_{ij}(\underline{X}) = a_{ij}(X_{ij})$. In order to incorporate some neighborhood information in the decision process, a neighborhood—the "context"—is defined, consisting of an arrangement of p pixels, such as shown in Fig. 1. The arrangement actually used will be based on physical and other practical considerations related to the environment and application. Let $\underline{X}_{ij} \in R^{np}$ be a p-vector of n-dimensional measurement vectors associated with pixel (i, j) to be classified and let $\underline{\theta}_{ij} \in \Omega^p$ be the corresponding p-vector of actual classes. The function $a_{ij}(\underline{X}_{ij})$ maps p vectors of observations into single classes (i.e., classifies pixel (i, j) based on $\underline{X}_{ij}$). The expected loss over the full array is

$$R(\underline{X}) = \frac{1}{N}\sum_{i,j} E\left[L(\theta_{ij}, a_{ij}(\underline{X}_{ij}))\right]. \tag{3}$$

Now if $L(\cdot, \cdot)$ is taken to be the 0-1 loss function (no loss for a correct classification, unit loss for an error) and the measurements in a neighborhood are assumed to be class conditionally independent, then the expected loss will be minimized if each pixel measurement X_{ij} is classified into class ω_k maximizing the *a posteriori* probability $p(\theta_{ij} | \underline{X}_{ij})$ or, equivalently, the discriminant function

$$g_k(\underline{X}_{ij}) = \sum_{\substack{\underline{\theta}_{ij} \in \Omega^p, \\ \theta_{ij} = \omega_k}} \left[\prod_{l=1}^{p} f(X_l | \theta_l)\right] G^p(\underline{\theta}_{ij}) \tag{4}$$

where

$X_l \in \underline{X}_{ij}$ measurement vector from the lth pixel in the p-array,

$\theta_l \in \underline{\theta}_{ij}$ class of the lth pixel in the p-array

$f(\underline{X}_l | \theta_l)$ class-conditional density of X_l given that the lth pixel is from class θ_l,

$G^p(\underline{\theta}_{ij}) = G^p(\theta_1, \theta_2, \cdots, \theta_p)$ *a priori* probability of observing the p-array $\theta_1, \theta_2, \cdots, \theta_p$.

Within the p-array, the pixel locations may be numbered in any convenient order. The joint probability distribution G^p is referred to as the *context distribution*.

To clarify the computation of the discriminant function defined by (4), consider the following example. Let the context array be the $p = 3$ choice shown in Fig. 1 with the pixels numbered such that the pixel (i, j) to be classified is associated with X_1 and θ_1, pixel $(i, j - 1)$ is associated with X_2 and θ_2, and pixel $(i - 1, j)$ is associated with X_3 and θ_3. Assume there are two possible classes: $\Omega = \{a, b\}$. Then, for instance, the discriminant function for class b is explicitly

$$g_b(\underline{X}_{ij}) = \sum_{\substack{\underline{\theta}_{ij} \in \Omega^3, \\ \theta_1 = b}} \left[\prod_{l=1}^{3} f(X_l | \theta_l)\right] G^3(\underline{\theta}_{ij})$$

$$= f(X_1 | b) f(X_2 | a) f(X_3 | a) G(b, a, a)$$
$$+ f(X_1 | b) f(X_2 | a) f(X_3 | b) G(b, a, b)$$
$$+ f(X_1 | b) f(X_2 | b) f(X_3 | a) G(b, b, a)$$
$$+ f(X_1 | b) f(X_2 | b) f(X_3 | b) G(b, b, b).$$

Note that $G^3(\underline{\theta}_{ij}) = G(\theta_1, \theta_2, \theta_3)$ is the relative frequency of occurrence in the scene of the specific configuration $(\theta_1, \theta_2, \theta_3)$.

An experiment was formulated to investigate the extent to which this classifier model can utilize contextual information in satellite-gathered remote sensing data. In order to avoid confounding other effects with the impact of context, it was decided to use a simulated data set generated as follows. A classification of multispectral remote sensing data was selected which had been judged to be very accurate (typically, produced by careful analysis and refinement of multitemporal data). Such a classification could be expected to embody the contextual content of an actual ground scene. Based on the classification map and using the associated statistics of the classes (developed in producing the classification) data vectors were produced by a Gaussian random number generator and composed into a new data set. Thus the new data set had the following characteristics.

1) Each pixel in the simulated data set represented the same class as in the "template" classification. The template could be considered the "ground truth" for the new data set.

2) All classes in the data set were known and represented.

3) All classes had multivariate Gaussian distributions with statistics typical of those found in real data.

4) All pixels were class-conditionally independent of adjacent pixels.

5) There were no mixture pixels.

Although the simulated data is somewhat of an idealization of "real" remote sensing data, its spatial organization is consistent with a real world scene and its overall characteristics are consistent with the context model set out above. In essence, then, what the experimental results based on the simulated data show is the effectiveness of the context classifier *given* that the underlying assumptions are reasonable. Further experiments are required to extend the conclusions of these results to real data.

Three data sets were selected to represent a variety of ground cover types and textures. Data set 1 is agricultural (Williston, ND), with ground resolution and spectral bands approximating those of the projected Landsat-D Thematic Mapper. Data set 2a is Landsat-1 data from an urban area (Grand Rapids, MI).

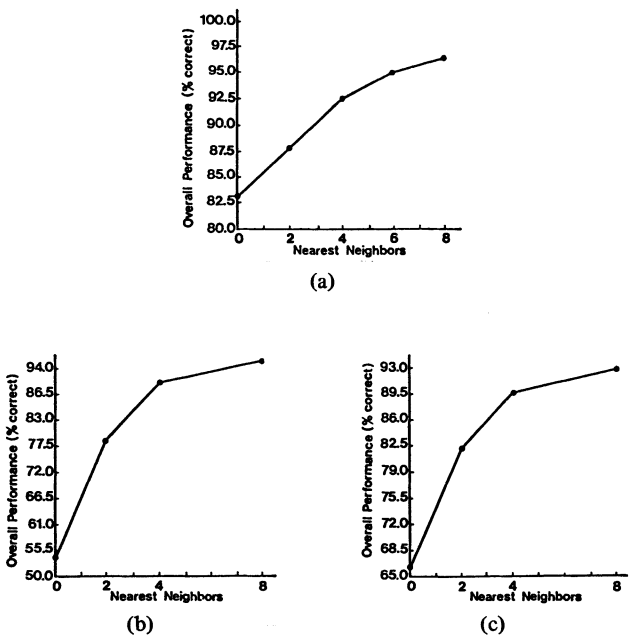

Fig. 2. Results for simulated data. (a) Data set 1. (b) Data set 2a. (c) Data set 2b.

Data set 2b is from the same Landsat frame as 2a, but from a locale having significantly different spatial organization. Each data set is square, 50 pixels on a side. Because of the resolution of the sensors involved, homogeneous areas in the data are small, typically a few tens of pixels in data set 1, rarely more than a few pixels in data sets 2a and 2b.

Fig. 2 shows the achieved classification results. The "no context" classification accuracy is plotted coincident with the vertical axis of each graph. Data set 1 was classified using successively 2, 4, 6, and 8 neighboring pixels; data sets 2a and 2b were classified using 2, 4, and 8 neighboring pixels. The results speak for themselves. The accuracy improvement resulting from the use of contextual information is quite significant.

For this experiment, the context distribution $G^p(\underline{\theta}_{ij})$ was simply tabulated from the "template" classification.[1] But in a real data situation, such a template is not available (else there would be no need to perform any further classification). One can envision a number of ways in which the p-vector distribution might be estimated for a remote sensing application. For example, it could be extracted from a classification of the same area obtained previously. This would require that the area not have changed too greatly in its class makeup since the earlier data collection and that the earlier classification was reasonably accurate. Or, the distribution might be obtained from a classification of any similarly constituted area. Still another possibility would be to estimate the p-vector distribution for the context classification from a "conventional" classification with "reasonably good" accuracy. All of these methods produce an *estimate* of the p-vector distribution, and a crucial question on which hinges the utility of this approach is how sensitive the contextual algorithm is likely to be to the "goodness" of the estimate. This question is the subject of current research which appears promising.

An experiment was formulated to obtain some evidence concerning the feasibility of applying the context classifier to a real data situation. The data set used covered a somewhat larger area of Grand Rapids, MI, containing both data sets 2a and 2b. Data from small areas of known ground cover were used to estimate the training class statistics, and data from a disjoint set of areas of known ground cover were used as "test samples" to evaluate the classifier accuracy (unfortunately, the set used for this test was rather small, consisting of only 136 pixels distributed among four urban classes).

A noncontextual classification was performed and found, based on the test set, to be 81.6-percent accurate. The p-vector distributions were estimated from this classification and used to perform contextual classifications using four and eight nearest neighbors. The four-neighbor classification was 83.1-percent accurate; the eight-neighbor classification was 84.6-percent accurate. For this case, then, some improvement in classification accuracy was achieved by incorporating context in the decision process, although the improvement was not as dramatic as for the simulated data sets. Whether this is due to poor estimation of the p-vector distributions or simply to less contextual information in the overall data set will be established by further investigation.

III. Multiprocessor Implementation of Classification Algorithms

A. Control Data Corporation (CDC) Flexible Processor System

Classification algorithms such as the context classifier (and even much simpler algorithms used for remote sensing data analysis) typically require large amounts of computation time. One way to reduce the execution time of these tasks is through the use of parallelism. Various parallel processing systems that can be used for remote sensing have been built or proposed. These include pipelined processors [6], multimicrocomputer systems [7], [8], and special purpose systems [9]. The CDC Flexible Processor System [6], [10], [11] is a commercially available multiprocessor system which has been recommended for use in remote sensing [12].

The basic components of a Flexible Processor are shown in Fig. 3. Each Flexible Processor is microprogrammed, allowing parallelism at the instruction level. An example of the way in which N Flexible Processors may be configured into a system is shown in Fig. 4. There can be up to 16 Flexible Processors linked together, providing much parallelism at the processor level. The clock cycle time of a Flexible Processor is 125 ns. Since 16 Flexible Processors can be connected in a parallel and/or pipelined fashion, the effective throughput can be drastically increased, resulting in a potential effective cycle time of less than 10 ns.

A central feature of the Flexible Processor is its dual 16-bit internal bus structure, enabling the Flexible Processor to

[1] As a practical matter, only the nonzero entries of the tabulated distribution are retained, stored as a binary tree structure. This represents a tradeoff between the amount of memory required to store the distribution (proportional to m^p for m possible classes) and the time required to access a given entry in computing the discriminant functions (which would be minimum if the entire distribution were stored in a p-dimensional array).

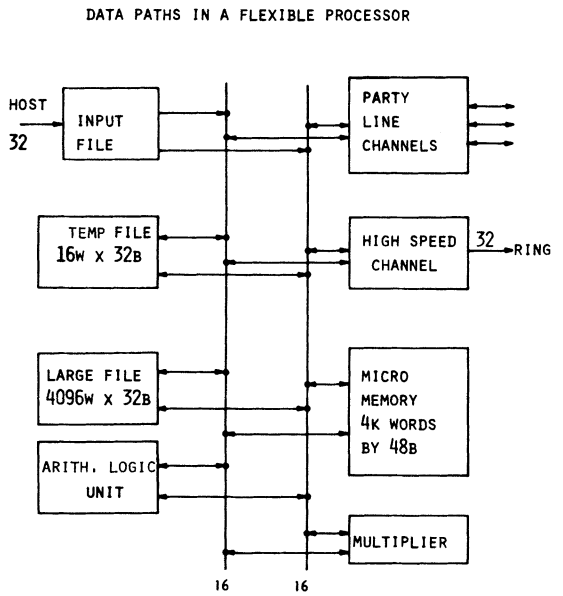

Fig. 3. Data path organization in the CDC Flexible Processor.

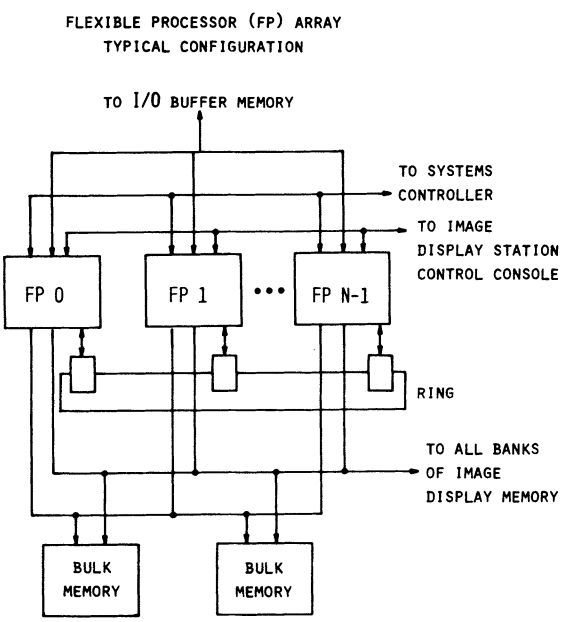

Fig. 4. Block diagram of typical Flexible Processor array.

manipulate either 16- or 32-bit operands. If 32-bit operands are used, the Flexible Processor can be programmed to execute floating point routines (on its integer hardware) based on the floating point representation of such systems as the IBM 370 and the PDP-11/70. If the needed data width is 16 bits, the Flexible Processor can be programmed to perform different operations on each of the 16-bit words simultaneously.

In each Flexible Processor, there are two files of registers, one called the temporary register file and the other the large register file. Both are divided into 16-bit subunits. If the needed path width is 16 bits, the two files can act like four files, thus creating more addressable user space. A special feature of the temporary file is its two separate read and two separate write address registers. This can save much CPU time in many types of matrix operations. The large register file has its own two read/write address registers. It is possible to do either a read or write to either file and simultaneously increment (or decrement) the address register. The temporary file is 16 words, 32 bits each, while the large file is 4096 words, 32 bits each. All of the register files consist of 60-ns random-access memory.

There are three 32-bit general purpose registers called the E, F, and G registers. All of these registers are connected to the arithmetic logic unit, which can perform 32-bit additions in 125 ns. The E and G registers are readable directly through the arithmetic logic unit. The general purpose registers can be shifted separately, or the E and F registers can be combined into a 64-bit shift register for double-length shifts. The output of the arithmetic logic unit is a 32-bit register that is addressable by byte (8 bits). This makes a variety of byte manipulations possible. Separate from the arithmetic logic unit is a hardware integer multiplier, which takes two bytes and multiplies them to produce a 16-bit result in 250 ns. The input registers are the P and Q registers, which are each 16 bits wide. The user can choose which two bytes are to be multiplied. The Flexible Processor is equipped with four index registers and eight corresponding compare registers. The index can be used for looping and can be incremented or decremented during any statement not addressing those registers. The Flexible Processor also contains a hardware jump stack, so it is capable of handling standard types of program calls such as subroutine jumps.

The micromemory consists of 4K 48-bit words. It stores the microprogram. Each Flexible Processor in a system can contain a different program.

Input/Output (I/O) for the Flexible Processor depends on the overall system (i.e., the Flexible Processor array and its host machine). A Flexible Processor is capable of interrupting another Flexible Processor for I/O. I/O among the Flexible Processors is done one of two ways. The first is a very high speed communication link, arranged in a ring configuration [10], [11]. Each Flexible Processor has a station on the ring, and each station on the ring is connected to two other stations, one to read and the other to write. When a Flexible Processor does a write to the ring, it gives 16 bits of data and the address of the destination. If a station receives data for another address, it shifts the data to the next station. This is continued until the data reaches the correct station. Special hardware has been added to remove data from the ring in the event of a station failure. Another form of I/O is through up to 16 64K banks of shared 160-ns memory. This is not as fast as the previous method; however for large data transfers, it frees the ring for other communications, as well as providing a buffer between Flexible Processors. Fig. 4 shows a typical Flexible Processor system configuration.

The Flexible Processor is programmed in "microassembly language," allowing parallelism at the instruction level. For example, it is possible to conditionally increment an index register, do a program jump, multiply two 8-bit integers, and add the E and G registers, all simultaneously. This type of operational overlap, in conjunction with the multiprocessing capability of the Flexible Processors, greatly increases the speed of the Flexible Processor array.

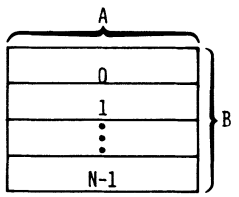

Fig. 5. An $A \times B$ image divided among N Flexible Processors.

In Section III-B, the use of a Flexible Processor array to perform maximum likelihood classifications is discussed. These ideas are extended for the contextual classifier in Section III-C.

B. Maximum Likelihood Classification on a Flexible Processor System

The pointwise maximum likelihood classification of pixels using a Flexible Processor array is discussed below. The contextual classifier presented in Section II performs computations similar to those used by the maximum likelihood classifier, but is complicated by the involvement of "neighboring" pixels. The analysis approach that has been taken is to first investigate the implementation of the maximum likelihood classifier and then extend the results to the contextual classifier.

Consider performing a maximum likelihood classification on an $A \times B$ image with N Flexible Processors. One way to approach the problem is to divide the image into N subimages and have each Flexible Processor perform the maximum likelihood classification for all pixels in its subimage. This is shown in Fig. 5. If all subimages have the same number of pixels, then the Flexible Processors will be fully utilized, i.e., on the average, the Flexible Processors will finish their computations simultaneously; no processor(s) will have to suspend operation to wait for others to finish. The classification of the entire image will take approximately $1/N$ as much time as it would take a single Flexible Processor to perform the entire classification. Thus maximum improvement, i.e., a factor of N, is obtained.

Consider the case where each subimage does not contain the same number of pixels, which will occur if $(A * B)/N$ is not an integer. This will lead to underutilization of the Flexible Processors, i.e., some processors will have to suspend operation to wait for others to finish. However, this underutilization will be negligible as will be shown.

One way to approach this situation is as follows. To each of N-1 Flexible Processors, assign a subimage of size.

$$\lceil (A * B)/N \rceil$$

where $\lceil x \rceil$, the ceiling of x, is the smallest integer greater than or equal to x. To the remaining Flexible Processor assign a subimage of size

$$(A * B) - (\lceil (A * B)/N \rceil * (N - 1)).$$

For example, if $A = 117$ and $B = 196$ (a typical LACIE image [13]) and $N = 16$, then

$$\lceil 22\,932/16 \rceil = \lceil 1433.25 \rceil = 1434$$

pixels are in each subimage associated with 15 Flexible Processors. The remaining pixels, of which there are

$$22\,932 - (15 * 1434) = 1422$$

are associated with one Flexible Processor. This sixteenth Flexible Processor will have fewer pixels to classify and thus will finish before the other Flexible Processors (assuming that, on the average, the time for the floating point calculations is approximately the same for all pixels), which implies some underutilization of this Flexible Processor. Ideally a factor of $N = 16$ performance improvement over a single Flexible Processor is desired, which, in this case, would require all 16 Flexible Processors to each classify 1434 pixels. To compute the utilization of the Flexible Processor array, divide the number of pixels actually classified by the maximum number that could be classified in the same amount of time if all 16 Flexible Processors were fully utilized. Thus the utilization is

$$22\,932/(16 * 1434) = 99+ \text{ percent}$$

Therefore, a factor of 99+ percent of N improvement is obtained.

In general, using the above assignment of pixels to subimages, the utilization of the system is

$$\frac{A * B}{\lceil (A * B)/N \rceil * N}.$$

The maximum value of the denominator is $A * B + N - 1$ and occurs when $A * B = k * N + 1$, where k is an arbitrary integer. Therefore,

$$\min ((A * B)/(\lceil (A * B)/N \rceil * N)) = (A * B)/(A * B + N - 1).$$

Based on typical sizes of remotely sensed images and the fact that the maximum size of a Flexible Processor array is 16,

$$A * B > 100 * N$$

and

$$(A * B)/(A * B + N - 1) > 99 \text{ percent}.$$

Thus, in general, the worst case performance is 99+ percent of the ideal factor of improvement over a single Flexible Processor.

The maximum likelihood classifier has been programmed on a simulator for a Flexible Processor array at LARS (Purdue's Laboratory for Applications of Remote Sensing). The simulator displays the contents of the main registers and provides a variety of tools for debugging Flexible Processor microcode. Preliminary tests indicate that a single Flexible Processor will perform a maximum likelihood classification faster than a PDP-11/70. Exact comparisons of the Flexible Processor array performance with other systems are difficult without detailed information about factors such as preprocessing and/or postprocessing of the data not included in the computation time, data precision used, memory load time, etc. The implementation of the maximum likelihood classifier currently running on the simulator indicates that an array of 16 Flexible Processors will require, on the average, approximately 11 s plus the time to access bulk memory to read pixel data in order to classify a 256 X 256 image into 16 classes. All of the

data values are assumed to be floating point with 16-bit mantissas and 7-bit exponents. Further optimization of this implementation is under study.

The experience gained through the use of the simulator has made evident the following advantages and disadvantages of the Flexible Processor system.

Advantages:

1) Multiple processors (up to 16);
2) user microprogrammable—parallelism at the instruction level;
3) connection ring for inter-Flexible Processor communications;
4) shared bulk memory units;
5) separate arithmetic logic unit and hardware multiply.

Disadvantages:

1) No floating point hardware;
2) microassembly language—difficult to program;
3) program memory limited to 4K microinstructions.

Based on the investigations to date, the advantages outweigh the disadvantages. However, alternative approaches, such as multimicroprocessor systems, must also be considered to determine the most cost-effective approach.

In the next subsection, the way in which a parallel processing system such as the Flexible Processor array can be used to perform context classification is examined.

C. Context Classification on a Flexible Processor System

Consider the implementation of a context classifier on an array of Flexible Processors. Assume the neighborhood is horizontally linear, as shown in Fig. 6. Divide the image into subimages of B/N rows A pixels long, as shown in Fig. 5. If $B = kN$, where k is an integer, there is 100-percent utilization of the Flexible Processors. Furthermore, there is no overhead for inter-Flexible Processor data transfers, since the entire neighborhood of each pixel is included in its subimage. Therefore, a factor of N improvement is attained.

If $(A * B)/N$ is an integer, but $B = kN + x$, $0 < x < N$, then Flexible Processors can be underutilized in order to keep neighborhoods within subimages, or Flexible Processors can be fully utilized, dividing neighborhoods between Flexible Processors, necessitating inter-Flexible Processor data transfers. This is shown for a simple example in Fig. 7, where $N = 2$, $A = 3$, and $B = 4$. In Fig. 7(a) no inter-Flexible Processor transfers are needed, but Flexible Processor 1 is not fully utilized. In Fig. 7(b) both Flexible Processors are fully utilized, but, due to the horizontally linear neighborhood, at least pixel 11 will have to be sent to Flexible Processor 1 and at least pixel 12 will have to be sent to Flexible Processor 0.

If $(A * B)/N$ is not an integer, some inter-Flexible Processor data transfers will be necessary. The number of transfers will be a function of the way in which the pixels are assigned to Flexible Processors, as in the previous paragraph. To determine the computationally fastest approach whenever $B = kN + x$, $0 < x < N$, requires knowledge of the actual image size, the actual number of Flexible Processors used, the exact time required to execute inter-Flexible Processor transfers, and the length of the neighborhood.

Fig. 6. Horizontally linear neighborhoods. Each box is 1 pixel.

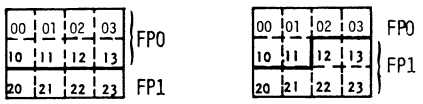

Fig. 7. Dividing an image into N subimages for horizontally linear neighborhoods, where $N = 2$, $A = 4$, and $B = 3$. (a) Underutilization, no inter-Flexible Processor data transfers required. (b) Inter-Flexible Processor data transfers required, full utilization.

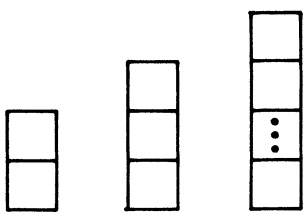

Fig. 8. Vertically linear neighborhood. Each box is 1 pixel.

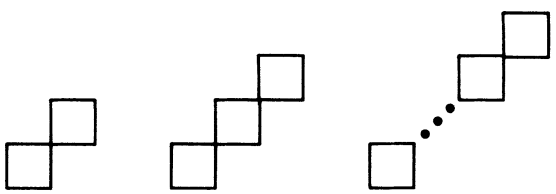

Fig. 9. Diagonally linear neighborhoods. Each box is 1 pixel.

There are two other cases of linear neighborhoods. These are vertically linear and diagonally linear, as shown in Figs. 8 and 9. The analysis for these two cases is similar to that for the horizontally linear case. The vertically linear case is just a 90° rotation of the horizontally linear case. The diagonally linear case can be simplified to a 45° rotation of the horizontally linear case for $B = kN$ by the proper assignment of pixels to Flexible Processors. Consider an $A \times B$ image, $A \leq B$, and $B = Nk$. Label the diagonals from 0 to $A + B - 2$, as shown in Fig. 10 for $A = 4$ and $B = 6$. The pixels can then be grouped into B sets of A pixels as follows.

1) For each i, $0 \leq i \leq A - 1$ the pixels in diagonals i and $i + B$ form a group of size A.

2) For each j, $A - 1 \leq j \leq B - 1$, the pixels in diagonal j form a group of size A.

Using these rules, each Flexible Processor is assigned k groups. Thus the problem has been reduced to the equivalent of the horizontally linear case, which has already been discussed. The case for $B = kN + x$, $0 < x < N$, is even more complex than for the analogous situation in the horizontally linear case, and requires a detailed tradeoff analysis based on the actual image size, the actual number of Flexible Processors used, the exact time required to execute inter-Flexible Processor data transfers, and the length of the neighborhood.

Now consider nonlinear neighborhoods, that is, neighborhoods which do not fit into one of the linear classes. For ex-

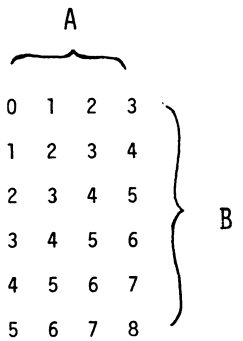

Fig. 10. The diagonals of an $A \times B$ image.

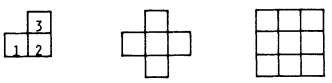

Fig. 11. Nonlinear neighborhoods. Each box is 1 pixel.

ample, all of the neighborhoods in Fig. 11 are nonlinear. Fig. 11(a) and its rotations represent the simplest nonlinear neighborhood. It is included in all other nonlinear neighborhoods. Thus that neighborhood is called the nonlinear kernel neighborhood.

It can be shown that there is no way to partition an $A \times B$ image into N (not necessarily equal) sections such that a context classifier using a nonlinear neighborhood can be performed without inter-Flexible Processor data transfers. This will be demonstrated for the nonlinear kernel, and will thus be true for all nonlinear neighborhoods. There are three cases to consider. If there is a horizontal border between two subimages stored in different Flexible Processors, then pixels 1 and 2 in Fig. 11(a) will be in different Flexible Processors. If there is a vertical border, pixels 2 and 3 will be in different Flexible Processors. If there is a diagonal border, pixels 1 and 2 will be in different Flexible Processors. The way in which to assign pixels to Flexible Processors in order to minimize computation time will depend upon the particular image size, number of Flexible Processors used, time required for inter-Flexible Processor communications and the shape and size of the neighborhood. These factors will also determine the effectiveness of the use of the Flexible Processor array for performing context classifications based on a given neighborhood.

IV. Conclusions

The preliminary results from the use of context in classification are promising. By studying ways of estimating the p-vector distribution, choosing the size and shape of neighborhood, etc., it may be possible to develop a highly accurate classifier for context-rich scenes.

The discussion of performing classifications with the Flexible Processor System demonstrates one way in which a multiple-processor system can be used to speed up the processing of image data. Future work involves programming the context classifier on the Flexible Processor simulator using different size and shape neighborhoods and determining the most efficient assignment of pixels to Flexible Processors for each case examined. The implementation of the classifier on the simulator and eventually on the actual Flexible Processor System will provide hard data to verify the effectiveness of the parallel processing approach.

Through the use of parallel, pipelined, and/or special purpose computer systems such as the CDC Flexible Processor System, the types of computations required for the context classifier and other computationally demanding processes can be implemented efficiently. This will not only reduce the computation time required to do contextual classification but will also allow the investigation of techniques which may otherwise be considered infeasible.

References

[1] P. H. Swain and S. M. Davis, Eds., *Remote Sensing: The Quantitative Approach*. New York: McGraw-Hill, 1978.

[2] R. M. Haralick, K. Shanmugam, and I. Dinstein, "Textural features for image classification," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-3, pp. 610–621, Nov. 1973.

[3] R. L. Kettig and D. A. Landgrebe, "Classification of multispectral image data by extraction and classification of homogeneous objects," *IEEE Trans. Geosci. Electron.*, vol. GE-14, pp. 19–26, Jan. 1976.

[4] J. S. Weszka, C. R. Dyer, and A. Rosenfeld, "A comparative study of texture measures and terrain classification," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-6, pp. 259–285, Apr. 1976.

[5] J. R. Welch and K. G. Salter, "A context algorithm for pattern recognition and image interpretation," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-1, pp. 24–30, Jan. 1971.

[6] G. R. Allen, L. O. Bonrud, J. J. Cosgrove, and R. M. Stone, "The design and use of special purpose processors for the machine processing of remotely sensed data," in *Proc. Conf. Machine Processing Remotely Sensed Data*, (IEEE Cat. No. 73CH0834-2GE), pp. 1A-25-1A-42, Oct. 1973.

[7] H. J. Siegel, "Preliminary design of a versatile parallel image processing system," in *Proc. Third Biennial Conf. Computing in Indiana*, pp. 11–25, (Indiana Univ., Bloomington, IN), Apr. 1978.

[8] H. J. Siegel, L. J. Siegel, R. J. McMillen, P. T. Mueller, Jr., and S. D. Smith, "An SIMD/MIMD multimicroprocessor system for image processing and pattern recognition," in *Proc. IEEE Computer Soc. Conf. Pattern Recognition and Image Processing*, (IEEE Cat. No. CH1428-2), pp. 214–224, Aug. 1979.

[9] K. S. Fu, "Special computer architectures for pattern recognition and image processing–an overview," in *Proc. 1978 National Computer Conf.*, pp. 1003–1013, June 1978.

[10] "Cyber-Ikon image processing system design concepts," Digital Image Syst. Div., Control Data Corp., Minneapolis, MN, Jan. 1977.

[11] "Cyber-Ikon flexible processor programming textbook," Digital Image Syst. Div., Control Data Corp., Minneapolis, MN, Nov. 1977.

[12] J. L. Kast, P. H. Swain, and T. L. Phillips, "The feasibility of using a Cyber-Ikon system as the nucleus of an experimental agricultural data center," Lab. for Applications of Remote Sensing, Purdue Univ., West Lafayette, IN. LARS Contract Rep. 021678, Feb. 1978.

[13] R. B. MacDonald, F. G. Hall, and R. B. Erb, "The use of Landsat data in a large area crop inventory experiment (LACIE)," in *Proc. Sym. Machine Processing of Remotely Sensed Data*, (IEEE Cat. No. 75CH1009-0-C), pp. 1B-1–1B-23, June 1975.