Reprinted from

# Symposium on Machine Processing of Remotely Sensed Data and Soil Information Systems and Remote Sensing and Soil Survey

**June 3-6, 1980**

# Proceedings


The Laboratory for Applications of Remote Sensing

Purdue University
West Lafayette
Indiana 47907 USA

IEEE Catalog No.
80CH1533-9 MPRSD

# PARALLEL PROCESSING IMPLEMENTATIONS OF A CONTEXTUAL CLASSIFIER FOR MULTISPECTRAL REMOTE SENSING DATA

HOWARD JAY SIEGEL, PHILIP H. SWAIN,
AND BRADLEY W. SMITH
Purdue University

ABSTRACT

Contextual classifiers are being developed as a method to exploit the spatial/spectral context of a pixel to achieve accurate classification. Classification algorithms such as the contextual classifier typically require large amounts of computation time. One way to reduce the execution time of these tasks is through the use of parallelism. The applicability of the CDC Flexible Processor system and of a proposed multimicroprocessor system (PASM) for implementing contextual classifiers is examined.

## I. INTRODUCTION

Contextual classifiers are being developed as a method to exploit the spatial/spectral context of a pixel to achieve accurate classification. Just as in written English one can expect to find certain letters occurring regularly in particular arrangements with other letters (qu, ee, est, tion), so certain classes of ground cover are likely to occur in the "context" of others. The former phenomenon has been used to improve character recognition accuracy in text-reading machines. We have demonstrated that the latter can be used to improve accuracy in classifying remote sensing data [1-3]. Intuitively this should not be surprising since one can easily think of ground cover classes more likely to occur in some contexts than in others. One does not expect to find wheat growing in the midst of a housing subdivision, for example. A close-grown, lush vegetative cover in such a location is more likely the turf of a lawn.

Classification algorithms such as the contextual 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. The Control Data Corporation Flexible Processor system is a commercially available multiprocessor system which has been recommended for use in remote sensing [4,5]. PASM is a proposed multimicroprocessor for image processing and pattern recognition [6].

Section II briefly describes the context classifier and gives an algorithm for performing it. The use of the Flexible Processor system to implement the classifier is explored in Section III. The use of PASM to implement the classifier is discussed in Section IV.

This work was sponsored in part by the National Aeronautics and Space Administration under Contract No. NAS9-15466.

CH1533-9/80/0000-0019 $00.75 ©1980 IEEE

## II. THE CONTEXTUAL CLASSIFIER

The image data to be classified are assumed to be a two-dimensional I-by-J array of multivariate pixels. Associated with the pixel at "row i" and "column j" is the multivariate measurement n- vector $X_{ij} \in R^n$ and the true class of the pixel $\theta_{ij} \in \Omega = \{\omega_1,\ldots,\omega_C\}$. The measurements have class-conditional densities $f(X|\omega_k)$, $k = 1,2,\ldots,C$, and are assumed to be class-conditionally independent. The objective is to classify the pixels in the array.

In order to incorporate contextual information into the classification process, when each pixel is to be classified p-1 of its neighbors are also examined. This neighborhood, including the pixel to be classified, will be referred to as the p-array. Intuitively, to classify each pixel, the contextual classifier computes the probability of the given observed pixel being in class k by also considering the measurement vectors (values) observed for the neighbor pixels in the p-array. Specifically, for each pixel, for each class in $\Omega$, a discriminant function g is calculated. The pixel is assigned to the class for which g is greatest. Each value of g is computed by summing the weighted probabilities of the p-1 neighbor pixels occurring in all possible classification states. This is described below mathematically for pixel (i,j) being in class $\omega_k$. The description is followed by an example to clarify the notation used. Further details may be found in [1,2,7].

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

where

$X_\ell \varepsilon \underline{X}_{ij}$ is the measurement vector from the $\ell$th pixel in the p-array (for pixel (i,j))

$\theta_\ell \varepsilon \underline{\theta}_{ij}$ is the class of the $\ell$th pixel in the p-array (for pixel (i,j))

$f(X_\ell | \theta_\ell)$ is the class-conditional density of $X_\ell$ given that the $\ell$th pixel is from class $\theta_\ell$

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

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

To clarify the computation of the discriminant function, consider the following example. Let the context array (neighborhood) be the p=3 choice shown in Figure II.1 with the pixels numbered such that the pixel (i,j) to be classified is associated with $X_1$ and $\theta_1$, pixel (i,j-1) is associated with $X_2$ and $\theta_2$, and pixel (i,j+1) is associated with $X_3$ and $\theta_3$. Assume there are two possible classes: $\Omega = \{a,b\}$. Then the discriminant function for class b is explicitly

$$g_b(\underline{X}_{ij}) = \sum_{\substack{\underline{\theta}_{ij} \varepsilon \Omega^3, \\ \theta_1 = b}} \left[ \prod_{\ell=1}^{3} f(X_\ell | \theta_\ell) \right] G^3(\underline{\theta}_{ij})$$

$$\begin{aligned} &= f(X_1|b)f(X_2|a)f(X_3|a)G(b,a,a) \\ &+ f(X_1|b)f(X_2|a)f(X_3|b)G(b,a,b) \\ &+ f(X_1|b)f(X_2|b)f(X_3|a)G(b,b,a) \\ &+ f(X_1|b)f(X_2|b)f(X_3|b)G(b,b,b) \end{aligned}$$

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 neighborhood configuration $(\theta_1, \theta_2, \theta_3)$.

After computing the discriminant functions $g_a$ and $g_b$ for pixel (i,j), pixel (i,j) is assigned to the class which has the larger discriminant function value.

Algorithm 1, shown in Figure II.2, is one way to implement the contextual classifier. The particular classifier considered here uses a horizontally linear p-array of size three. This is shown in Figure II.1.

First consider the main loop. Let the original image to be classified be an I-by-J array called A. Columns 0 and J-1, the two side edges of the image, are not classified since these pixels will not have both right and left neighbors. The variable "value" will contain the maximum "g" (discriminant function) value calculated for pixel (i,j). This variable may be updated as the "g" for each class is calculated. The variable "class" is the class associated with "value." In the main loop, "g(i,j,k)" is a call to a function to calculate the discriminant function for pixel (i,j) and class k. This function is called I * (J-2) * C times, once for each class for each pixel being classified.

Consider the calculation of g(i,j,k). The class of pixel (i,j) is held constant at k, while all other possible class assignments are considered for pixels (i,j-1) and (i,j+1). For each assignment of classes for the pixels neighboring pixel (i,j), of which there are C*C, the product of the class-conditional densities ("compf") is weighted by "G(r,k,q)," the a priori probability of observing the 3-array $(\omega_r, \omega_k, \omega_q)$. The "G" array is predetermined and prestored. For each call "g(i,j,k)," the value of "sum" for that i,j, and k is calculated. "Sum" is then returned as the value of "g(i,j,k)." In this straightforward version of the g(i,j,k) routine, the function to compute a class-conditional density ("compf") is called C*C times each time "g" is called.

Now consider the "compf" routine. This calculates the class-conditional density for pixel (a,b) and class k using the following equation:

$$f(x|k)=e^{\log|\Sigma_k| - \left[(x-m_k)^T \Sigma_k^{-1}(x-m_k)\right]/2}$$

where the measurement vector for each pixel is of size four, $\Sigma_k^{-1}$ is the inverse covariance matrix for class k (four-by-four matrix), $m_k$ is the mean vector for class k (size four vector), "T" indicates the transpose, and "log" is the natural logarithm. For each class, the algorithm uses $\log|\Sigma_k|$, $\Sigma_k^{-1}$, and $m_k$ as precomputed constants. For each call "compf (a,b,k)," the value of "$e^{expo}$" for that a,b, and k is calculated. "$e^{expo}$" is then returned as the value of "compf(a,b,k)."

Algorithm 1 executes the "compf" subroutine $I*(J-2)*C^3$ times. Since for each pixel there are C "f"s (class-conditional densities), this approach is inefficient by a factor of $C^2$. Algorithm 2 rectifies this problem by saving certain "f" values rather than recalculating them.

The Algorithm 2, shown in Figure II.3, implements the contextual classifier without the redundant executions of "compf" that occur in Algorithm 1. Let X, Y, and Z correspond to the pixels (i,j-1), (i,j), and (i,j+1), respectively, where (i,j) is the pixel to be classified. Each of X, Y, and Z is a vector of size C. Element t of X will contain the class-conditional density ("compf") for the current (i,j-1) pixel for class t. Y and Z are defined similarly. By using these three vectors to save the class-conditional densities, each density (for a given pixel and class) is calculated only once, instead of $C^2$ times.

The main loop of Algorithm 2 is modified to calculate the class-conditional densities for the first three columns each time a new row is considered (i.e., each time "i" is incremented). Each time a new pixel in a given row is to be classified (i.e., just before "j" is incremented), these values are updated. In particular, X gets the Y values, Y gets the Z values, and new values are calculated to update Z.

The new discriminant function calculation, g', does not call the subroutine "compf." It gets the values it needs from the X, Y, and Z arrays. For each call "g'(k)," the value of "sum" for that k is calculated. "Sum" is then returned as the value of "g'(k)."

The same "compf" routine is used for both Algorithms 1 and 2. Algorithm 1 calls this routine $I*(J-2)*C^3$ times, while Algorithm 2 calls it only $I*(J-2)*C$ times.

There are other techniques that can be employed to make Algorithm 2 even more efficient that have not been included in order to avoid obscuring the basic program flow.

The serial complexity of Algorithm 2 can be calculated in terms of assignment statements, multiplications, additions, and "compf" calculations. To initialize X, Y, and Z for new rows, I*C*3 assignments and calls to "compf" occur. For each pixel, at most C+1 assignments to "value" and "class" occur, C assignments to "current" occur, and C calls to "g'(k)" occur. In addition, for each row, the X, Y, and Z vectors are updated J-3 times, each update using 3*C assignments and C calls to "compf." Each execution of "g'(k)" uses $3*C^2$ multiplications, $C^2$ additions, and $C^2+1$ assignments. Thus, the total complexity for Algorithm 2 is:

| | |
|---|---|
| $I(J(C^3+7C+2)-(2C^3+14C+4)$ | assignments; |
| $3C^3I(J-2)$ | multiplications; |
| $C^3I(J-2)$ | additions; and |
| $I*J*C$ | "compf" calculations. |

The growth is proportional to $I*J*C^3$ assignments, multiplications and additions, and I*J*C "compf" calculations.

In this section, a contextual classifier based on a horizontally linear neighborhood of size three has been analyzed. Algorithms for contextual classifiers using other size and shape neighborhoods would be analogous to the algorithms which were presented.

Algorithms 1 and 2 are written for conventional uniprocessor systems. Sections III and IV will examine how to implement Algorithm 2 on a CDC Flexible Processor system and on a multimicroprocessor system such as PASM.

## III. FLEXIBLE PROCESSOR SYSTEM IMPLEMENTATION OF THE CONTEXTUAL CLASSIFIER

This section discusses programming a CDC Flexible Processor system [4] simulator to perform a size three linear neighborhood contextual classifier. The Flexible Processor system is briefly overviewed. Then the simulation is described.

The basic components of a Flexible Processor (FP) are shown in Figure III.1. Each FP is microprogrammed, permitting parallelism at the instruction level. An example of the way in which N FPs may be configured into a system is shown in Figure III.2. There can be up to 16 FPs linked together, providing much parallelism at the processor level. The FPs can communicate among themselves through the high-speed ring or shared bulk memory. The clock cycle time of each FP is 125 nsec (nanoseconds). Since 16 FPs 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 nsec.

An FP is programmed in micro-assembly 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 two 32-bit integers -- all simultaneously. This type of operational overlap, in conjunction with the multiprocessing capability of the FPs, greatly increases the speed of the FP array.

The following list summarizes the important architectural features of an FP:

User microprogrammable.
Dual 16-bit internal bus system.
Able to operate with either 16- or 32-bit words.
125 nsec clock cycle.
125 nsec time to add two 32-bit integers.
250 nsec time to multiply two 8-bit integers.
Register file (with 60 nsec access time) of over 8,000 16-bit words.

In order to debug, verify, and time FP algorithms, a simulator for an array of up to 16 FPs has been developed. This simulator runs under the UNIX operating system on a PDP-11 series computer at LARS and has been used to program a maximum likelihood classifier [1]. An assembler for the micro-assembly language has also been developed.

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

Advantages:

Multiple processors (up to 16).
User microprogrammable -- parallelism at the instruction level.
Connection ring for inter-FP communications.
Shared bulk memory units.
Separate arithmetic logic unit and hardware multiply.

Disadvantages:

No floating-point hardware.
Micro-assembly language -- difficult to program.
Program memory limited to 4k microinstructions.

More details about the FP may be found in [8]. Information about the assembler and simulator used at LARS to assemble and execute the FP programs for the contextual classifier is presented in [7].

Consider the implementation of a contextual classifier on an array of N FPs. Assume the neighborhood is horizontally linear, as shown in Figure III.3. Divide the A-by-B image into subimages of B/N rows A pixels long, as shown in Figure III.4. Assign each subimage to a different FP. The entire neighborhood of each pixel is included in its subimage. Each FP can therefore execute the uniprocessor algorithm presented in Section II on its own subimage. No interaction between FPs is needed, i.e., each FP can process its subimage independently.

The LARS FP microassembler and simulator are being used to gather statistics on the execution time for the size three horizontally linear neighborhood contextual classifier. Due to the fact that each FP is microprogrammable, determining program correctness and analyzing execution times is done through the use of the microassembler and simulator. The current implementation of the contextual classifier uses 744 microinstructions, stored in the micromemory (see Figure III.1). The format of the data words of the pixel measurement vectors, covariance matrices, etc., consists of a 14-bit two's complement exponent and a 17-bit sign-magnitude mantissa. The covariance matrices, logarithms of the determinants of the covariance matrices, a priori probabilities ($G^P$), and the X, Y, and Z vectors are all stored in the large file (see Figure III.1). In this way, each FP has all the information it needs for performing the classification on its subimage. The subimage data itself would be stored in a bulk memory (see Figure III.2). A multiple FP configuration which associates one bulk memory with each FP would be best for this application. For testing the FP contextual classifier program, the classification of one row of eight pixel measurement vectors (stored in the large file) using four classes is being evaluated. The FP contextual classifier program is currently being debugged. The timing results of using the FP simulator to classify actual data using Algorithm 2 (Figure II.3) will be presented at the symposium.

For the horizontally linear neighborhoods, when using N FPs together to process an image, each FP handles 1/N-th of the image. Therefore, nearly a factor of N improvement is attained over the time required for one FP to implement the contextual classifier. (A perfect factor of N improvement occurs if B is a multiple of N. The minor degradation in performance when B is not a multiple of N is discussed in [2].) Vertically linear and diagonally linear neighborhoods (Figure III.5) can be processed in a manner similar to that for horizontally linear neighborhoods [2].

Consider nonlinear neighborhoods, that is, neighborhoods which do not fit into one of the linear classes. For example, all of the neighborhoods in Figure III.6 are nonlinear. It can be shown that there is no way to partition an image into N (not necessarily equal) sections such that a contextual classifier using a nonlinear neighborhood can be performed without data transfers among FPs [2]. The way in which to assign pixels to FPs in order to minimize computation time will depend upon the particular image size, number of FPs used, the time required for inter-FP communications, and the shape and size of the neighborhood. A detailed analysis of the interaction of these factors is currently under study.

## IV. MULTIMICROPROCESSOR IMPLEMENTATION OF THE CONTEXTUAL CLASSIFIER

This section describes a method for implementing the contextual classifier on a large-scale multimicroprocessor system such as PASM [6,9-11]. PASM is a dynamically reconfigurable system being designed at Purdue University for image processing and pattern recognition tasks. The PASM design will support up to 1024 processors. Other computer architects have proposed parallel processing systems with $2^{14}$ to $2^{16}$ microprocessors [12,13]. The method for implementing the contextual classifier on PASM will be based on the use of the SIMD mode of parallelism.

The acronym SIMD stands for "single instruction stream -- multiple data stream" [14]. Typically, an SIMD machine is a computer system consisting of a control unit, N processors, N memory modules, and an interconnection network. The control unit broadcasts instructions to all of the processors, and all active processors execute the same instruction at the same time. Thus, there is a single instruction stream. Each active processor executes the instruction on data in its own associated memory module. Thus, there is a multiple data stream. The interconnection network, sometimes referred to as an alignment or permutation network, provides a communications facility for the processors and memory modules. Examples of existing SIMD machines include the Illiac IV and STARAN [15,16].

One way to model the physical structure of an SIMD machine is shown in Figure IV.1. As indicated, there are N processing elements (PEs) where each PE consists of a processor with its own memory. The PEs receive their instructions from the control unit and communicate through the interconnection network.

To demonstrate how SIMD machines operate, consider the following simple task. Assume that A, B, and C are each one-dimensional arrays (vectors) and that the task to be performed is the elementwise addition of A and B, storing the result in C. In a uniprocessor system, this can be expressed as:

<u>for</u> i = 0 <u>to</u> N-1 <u>do</u>

C(i) = A(i) + B(i)

This computation will take N steps on a serial machine.

Assume that A, B, and C are stored in a SIMD machine, with N PEs, such that A(i), B(i), and C(i) are all stored in the memory of PE i, $0 \leq i < N$. To perform an elementwise addition of the vectors A and B and store the result in C, all PEs would execute (simultaneously)

C = A + B

with PE i doing the addition of A(i) and B(i), storing the result in C(i). Thus, in this case, the SIMD machine does in one step a task requiring N steps on a serial processor.

Consider a variation on this example. Assume the N-step serial task is:

<u>for</u> i = 1 <u>to</u> N-1 <u>do</u>

$$C(i) = A(i) + B(i-1)$$

$$C(0) = A(0)$$

Given the data allocation above (i.e., A(i), B(i), and C(i) in PE(i)), an SIMD machine does this task in three different steps:

1. The value of B(i-1) is moved, through the interconnection network, from PE i-1 to PE i, $1 \leq i < N$. Most proposed and existing SIMD interconnection networks can do this in one parallel data transfer [17].

2. In PE i, add A(i) to B(i-1) and store the result in C(i), $1 \leq i < N$ (PE 0 is disabled).

3. In PE 0, store A(0) in C(0) (all other PEs are disabled).

Thus, this example demonstrates the need for the interconnection network and methods for disabling PEs.

This simple example was provided to familiarize the reader with the concept of the SIMD mode of parallel processing. More complex examples involving image processing and feature extraction can be found in [18,19].

Consider the implementation of the contextual classifier discussed in Sections II and III on a microprocessor-based SIMD machine. Recall that the neighborhood is as shown in Figure II.1, i.e., a horizontally linear neighborhood with p=3. The approach to decomposing the task will be similar to that used in Section III for the FP system. In both cases, the image is divided into N subimages, and each subimage is assigned to a different processor for classification computations. However, there are three main differences:

1. It is technologically and economically feasible to construct a multimicroprocessor SIMD machine with many more than 16 processors. Therefore, while the "N" for the FP system is limited by 16, the "N" for the multimicroprocessor system could be as large as 256, 512, or 1024.

2. The differences in computational capabilities between an FP and an off-the-shelf microprocessor must be considered. For example, depending on the microprocessor chosen, 16 FPs may be faster than 32 microprocessors.

3. In the SIMD mode of parallelism, the program (Algorithm 2) is stored in the control unit, not in each microprocessor. The control unit broadcasts the instructions to the microprocessors. The control unit would also store the $G^p$ array, broadcasting the appropriate array element to all the microprocessors when it is needed. In the FP system, each FP would store a copy of the program and must store or have access to the $G^p$ array.

Thus, a SIMD machine can be used to perform the contextual classification based on a horizontally linear neighborhood of size three without any inter-PE communication. As in the case of using the FP system to implement the classifier, the implementation using an SIMD machine with N microprocessors can achieve as much as a factor of N improvement over the use of a single microprocessor. The exact improvement will be a function of the image size and N.

To attain a perfect factor of N improvement, B (in Figure III.4) would have to be a multiple of N. Since N in the SIMD case would be a multiple of the N in the FP case, this is less likely to occur. When B is not a multiple of N, then (a) some PEs may have to process more rows than others (leaving some PEs underutilized), or (b) each PE would process a subimage including a partial row (requiring inter-PE data transfers). The alternative which is best would depend on the image size, the way in which subimages are allocated to PEs, N, the processor speed, and the interconnection network speed. The situation for vertically linear and diagonally linear neighborhoods is similar. Nonlinear neighborhoods require inter-PE communications, but the best way to implement such a classifier would depend on the factors just mentioned and the neighborhood size and shape. These implementation considerations are currently being explored.

## V. CONCLUSIONS

Algorithms for performing contextual classifications using a size three horizontally linear neighborhood were presented. Algorithm 1 was a straightforward approach. Algorithm 2 was a more efficient approach that avoided redundant calculations. The serial computational complexity of Algorithm 2 was shown to have a growth proportional to $I*J*C^3$ assignments, multiplications, and additions, and I*J*C "compf" calculations. The way in which N FPs could perform the classifications N times faster than a single FP was explained. The use of N microprocessors in the SIMD mode of parallel processing to do the classifications N times faster than a single microprocessor was discussed.

In summary, contextual classifiers have been shown to be powerful remote sensing tools in other papers. Their main disadvantage is their computation complexity. This paper has demonstrated how parallel processing can be used to overcome this disadvantage.

## VI. REFERENCES


1. P. H. Swain, H. J. Siegel, and B. W. Smith, "A method for classifying multispectral remote sensing data using context," Proceedings of the 1979 Machine Processing of Remotely Sensed Data Symposium (IEEE Catalog No. 79 CH 1430-8 MPRSD), pp. 343-353, June 1979.

2. P. H. Swain, H. J. Siegel, and B. W. Smith, "Contextual classification of multispectral remote sensing data using a multiprocessor system," scheduled to appear IEEE Transactions on Geoscience Electronics, April 1980.

3. J. C. Tilton, P. H. Swain, and S. B. Vardeman, "Context distribution estimation for contextual classification of multispectral image data," in these proceedings.

4. Control Data Corp., Cyber-Ikon Image Processing System Design Concepts, Digital Systems Division, Control Data Corp., Minneapolis, MN, January 1977.

5. 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, LARS Contract Report 021678, Laboratory for Applications of Remote Sensing (LARS), Purdue University, West Lafayette, IN, February 1978.

6. 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," Proceedings of the 1979 IEEE Computer Society Conference on Pattern Recognition and Image Processing (IEEE Catalog No. 79 CH 1428-2C), pp. 214-224, August 1979.

7. P. H. Swain, P. E. Anuta, D. A. Landgrebe, and H. J. Siegel, Vol. III: Processing Techniques Development, Part 2: Data Preprocessing and Information Extraction Techniques, LARS Contract Report 113079, Laboratory for Applications of Remote Sensing (LARS), Purdue University, West Lafayette, IN, November 1979.

8. Control Data Corp., Cyber-Ikon Flexible Processor Programming Textbook, Digital System Division, Control Data Corp., Minneapolis, MN, November 1977.

9. H. J. Siegel, "Preliminary design of a versatile parallel image processing system," Proceedings of the Third Biennial Conference on Computing in Indiana, Indiana University, Bloomington, IN, April 1978.

10. H. J. Siegel, P. T. Mueller, Jr., and H. E. Smalley, Jr., "Control of a partitionable multimicroprocessor system," Proceedings of the 1978 International Conference on Parallel Processing (IEEE Catalog No. 78 CH 1321-9C), pp. 9-17, August 1978.

11. H. J. Siegel, F. Kemmerer, and M. Washburn, "Parallel memory system for a partitionable SIMD/MIMD machine," Proceedings of the 1979 International Conference on Parallel Processing (IEEE Caralog No. 79 CH 1433-2C), pp. 212-221, August 1979.

12. H. Sullivan, T. R. Bashkow, and D. Klappholz, "A large-scale homogeneous, fully distributed parallel machine," Proceedings of the Fourth Annual Symposium on Computer Architecture (IEEE Catalog No. 77 Ch 1182-5C), pp. 105-124, March 1977.

13. M. C. Pease, "The indirect binary n-cube microprocessor array," IEEE Transactions on Computers, Vol. C-26, No. 5, pp. 458-473, May 1977.

14. M. J. Flynn, "Very high-speed computing systems," Proceedings of the IEEE, Vol. 54, pp. 1901-1909, December 1966.

15. W. J. Bouknight, et al, "The Illiac IV system," Proceedings of the IEEE, Vol. 60, pp. 369-388, April 1972.

16. K. E. Batcher, "STARAN parallel processor system hardware," AFIPS Conference Proceedings Volume 43: 1974 National Computer Conference, pp. 405-410, May 1974.

17. H. J. Siegel, "Interconnection networks for SIMD machines," Computer, Vol. 12, No. 6, pp. 57-65, June 1979.

18. L. J. Siegel, P. T. Mueller, Jr., and H. J. Siegel, "FFT algorithms for SIMD machines," Proceedings of the Seventeenth Annual Allerton Conference on Communications, Control, and Computing, pp. 1006-1015, University of Illinois - Urbana, October 1979.

19. H. J. Siegel, P. H. Swain, L. J. Siegel, P. T. Mueller, Jr., and J. El-Achkar, Parallel Image Processing/Feature Extraction Algorithms and Architecture Emulation, Technical Report TR-EE 79-51, School of Electrical Engineering, Purdue University, West Lafayette, IN, November 1979.

Main Loop

```
for i = 0 to I-1 do /* row */
  begin
  for j = 1 to J-2 do /* column */
    begin               /* for each pixel */
    value = -1          /* max "g" */
    class = -1          /* class with max
                                      "g" */
    for k = 1 to C do /* for each class */
      begin
      current = g(i,j,k)
      if current > value
        then  value = current
              class = k
      end
    print Pixel (i,j) is classified as
         "class"
    end
  end
```

Figure II.2. Algorithm 1 -- Implementation of a contextual classifier. Main Loop.

Discriminant Function Calculation

```
function        g(i,j,k)
sum = 0
for r = 1 to C do  /* all possible
                              classes */
  begin
  for q = 1 to C do  /* all possible
                              classes */
    begin
    sum = compf(i,j-1,r)*compf(i,j,k)
          *compf(i,j+1,q)*G(r,k,q)+sum
    end
  end
return (sum)
```

Class-Conditional Density Calculation

```
function compf(a,b,k) /* for pixel (a,b),
                                 class k */
x = A(a,b)    /* x is pixel measurement
                                 vector */
```

expo = $\log|\Sigma_k| - \left[(x-m_k)^T \Sigma_k^{-1} (x-m_k)\right] * .5$

return $(e^{expo})$

Figure II.2 (cont.). Algorithm 1 -- Discriminant function and class-conditional density routines.

Main Loop

```
for i = 0 to I-1 do  /* row */
  begin
  for k = 1 to C do
    begin /* compute f's for 1st 3
                               columns */
    X(k) = compf (i,0,k)
    Y(k) = compf (i,1,k)
    Z(k) = compf (i,2,k)
    end
  for j = 1 to J-2 do /* column */
    begin        /* for each pixel */
    value = -1   /* max "g" */
    class = -1   /* class with max "g" */
    for k = 1 to C do
        begin
        current = g'(k)
        if current > value
          then value = current
               class = k
        end
    print Pixel (i,j) is classified as
                                "class"
    if j < J-2
        then /* update X,Y,Z arrays */
             for k = 1 to C do
               begin
               X(k) = Y(k)
               Y(k) = Z(k)
               Z(k) = compf (i,j+2,k)
               end
    end
  end
```

Figure II.3. Algorithm 2 -- Implementation of a contextual classifier. Main Loop.

Discriminant Function Calculation

```
function  g'(k)
sum = 0
for r = 1 to C do  /* all possible
                          classes */
  begin
  for q = 1 to C do /* all possible
                          classes */
    begin
    sum = X(r) * Y(k) * Z(q)
          *G(r,k,q) + sum
    end
  end
return (sum)
```

Figure II.3 (cont.). Algorithm 2 -- Discriminant function calculation.

| | | |
|---|---|---|
| (i,j-1) | (i,j) | (i,j+1) |

Figure II.1. A p=3 context array (neighborhood).

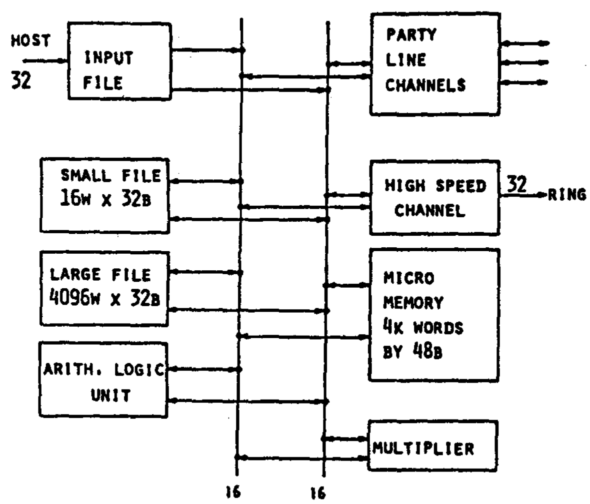


Figure III.1. Data path organization in the CDC Flexible Processor.

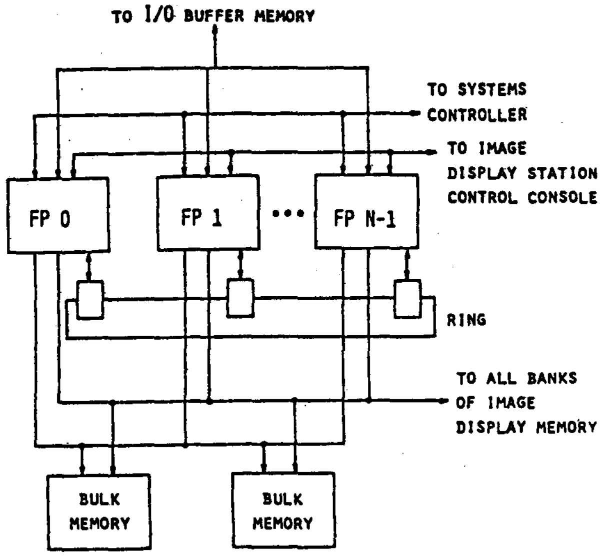


Figure III.2. Block diagram of typical Flexible Processor array.

Figure III.3. Horizontally linear neighborhoods. Each box is one pixel.

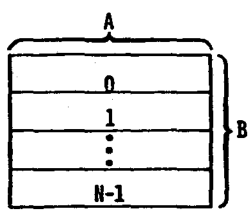


Figure III.4. An A-by-B image divided among N Flexible Processors.

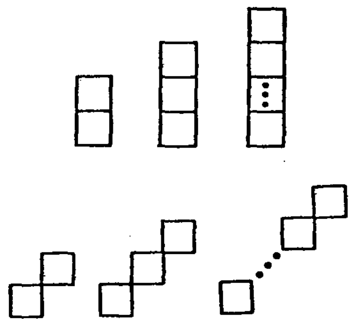

Figure III.5. Vertically linear and diagonally linear neighborhoods. Each box is one pixel.

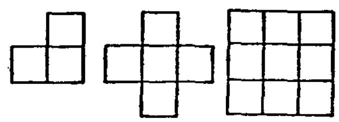

Figure III.6. Nonlinear neighborhoods. Each box is one pixel.

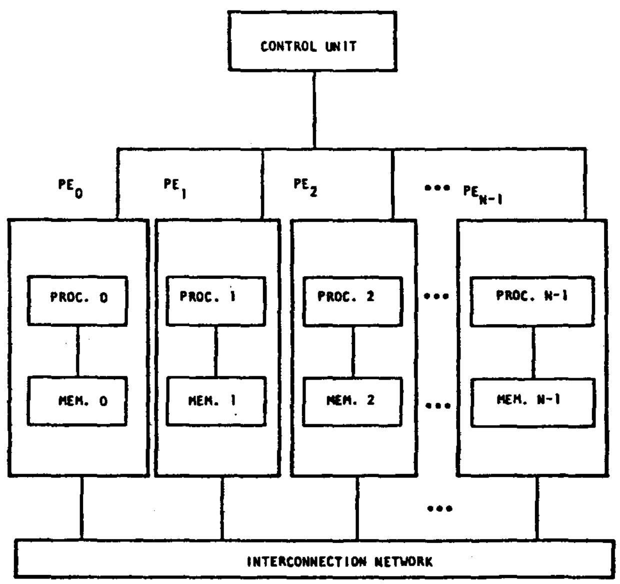


Figure IV.1. A general model of an SIMD machine.

Howard Jay Siegel is an assistant professor in the School of Electrical Engineering at Purdue University and is on the research staff at LARS. His research interests include parallel processing, multimicroprocessor systems, speech processing, and image processing. He received both a B.S.E.E. and a B.S. degree in Management from MIT and the M.S., M.S.E., and Ph.D. degrees from the Department of Electrical Engineering and Computer Science at Princeton. He is a member of the Eta Kappa Nu and Sigma Xi honorary societies. He is currently vice-chairman of ACM SIGARCH (Special Interest Group on Computer Architecture) and an IEEE Computer Society Distinguished Visitor.

Bradley W. Smith received the B.S. degree in December 1978 from the School of Electrical Engineering at Purdue University. He is currently pursuing the M.S.E.E. degree at Purdue and is on the research staff of the University's Laboratory for Applications of Remote Sensing (LARS). His research interests include computer architecture, parallel processing, and remote sensing. He is a member of the engineering honorary Tau Beta Pi and the science honorary Phi Kappa Phi.

Philip H. Swain is assistant professor of electrical engineering, Purdue University, and program leader for Data Processing and Analysis Research at the University's Laboratory for Applications of Remote Sensing (LARS); B.S.E.E., Lehigh University; M.S.E.E. and Ph.D., Purdue University. Prof. Swain has been affiliated with LARS since 1966 and has contributed extensively to the development of data processing methods for the management and analysis of remote sensing data. His areas of specialization include theoretical and applied pattern recognition and methods of artificial intelligence. He is co-editor and contributing author for the textbook Remote Sensing: The Quantitative Approach (McGraw-Hill, 1978).