Reprinted from

Eighth International Symposium

Machine Processing of

Remotely Sensed Data

with special emphasis on

Crop Inventory and Monitoring

July 7-9, 1982

Proceedings

Purdue University
The Laboratory for Applications of Remote Sensing
West Lafayette, Indiana 47907 USA

PARALLEL PROCESSING CONCEPTS FOR REMOTE SENSING APPLICATIONS

B.W. SMITH, H.J. SIEGEL, P.H. SWAIN

Purdue University/School of Electrical Engineering
West Lafayette, Indiana

ABSTRACT

Previous research has suggested a specific SIMD (Single Instruction stream - Multiple Data stream) machine architecture for the application of parallel processing to remote sensing tasks. It is a large-scale multimicroprocessor structure which could consist of as many as 1024 processors. This type of architecture is extremely well-suited to the execution of window-based (e.g., image correlation) and pixel-based (e.g., maximum likelihood classification) types of operations. The analysis of a number of remote sensing data processing techniques for implementation on a machine with this architecture are discussed. This includes both the design of parallel algorithms and the exploitation of appropriate data structures.

I. INTRODUCTION

Multispectral image data collected by remote sensing devices aboard aircraft and spacecraft are relatively complex data entities. Because of its multispectral nature, vectors are used to represent the data. The execution of even the simplest classification algorithms may require large amounts of computation time. Thus, in order to allow complex classification algorithms to become more feasible, special hardware to increase the execution speed is of interest.

Through the use of parallelism, the execution time of classification algorithms can be reduced. There are several classes of parallel processing systems. An SIMD (Single Instruction stream - Multiple Data stream) [1] machine typically consists 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 (enabled) processors execute the same instruction at the same time. Each active processor executes the broadcast instruction on data in its own associated memory module. The interconnection network provides a communications facility for the processors and memory modules. An MIMD (Multiple Instruction stream - Multiple Data stream) [1] machine typically consists of N processors and N memories, where each processor can follow an independent instruction stream. As with SIMD architecture, there is a multiple data stream and an interconnection network.

For many remote sensing tasks, all pixels in a given image are treated in a similar fashion. This implies that the same numerical operations are done on all pixels. Thus, the same instructions are performed on multiple data sets. It would appear that SIMD machines are particularly well-suited to these tasks. Further, since images can be as large as 65,536 pixels on an edge, a system that has as many as 1024 processors would also be well-suited for image processing tasks. Large scale integration makes just such parallel systems possible.

The applications of such a machine to image processing tasks is the topic under consideration here. Section II discusses a potential machine architecture. Sections III, IV, V, and VI discuss how such a system can be applied to smoothing, maximum likelihood classification, contextual classification, and image correlation, respectively.

II. MACHINE ARCHITECTURE

The proposed SIMD architecture, *Mu*ltimicroprocessor *R*emote *S*ensing *S*ystem (*MuRSS*) is shown in Fig. 1. The system consists of N processing units (*PUs*) numbered from 0 to N-1 and 2N+1 memory modules numbered from 0 to 2N. Each PU will be a commonly available microprocessor, such as a 68000 [2] equipped with a floating point unit. Four busses will be connected to each PU. One bus to each will be used to communicate with the control unit, while the remaining three busses will be connected to as many as 2^8 64K-byte banks of memory. Two of the three busses, and consequently the associated memory banks, will also be connected to adjacent PUs. Thus, the memory banks that are "shared" can be used to store common data. This eliminates the need for a more complex interconnection structure. Memory contention is not a problem. The only way contention can occur is if two PUs try to access the same memory. This can not happen with this SIMD system, since whenever PU i is using its 0 bus, PU i-1 must also be using its 0 bus (it cannot be using its 2 bus). The multiple banks of memory will allow the host to load/unload data into/from half the banks, while the PU operates on data from the other half, maximizing overlap.

The control unit (*CU*) will be a special purpose processor. It will be equipped with memory, in which it will store its program, global data, the program to be broadcast to the PU, and its local variables. The amount of memory is variable and is a function of cost and the processor chosen for the CU.

The host will be assumed to be a computer such as an IBM-370 or a PDP-11 series machine. All support operations, such as formating input and formating output, will be performed by the host.

III. SMOOTHING ON A PARALLEL SIMD MACHINE.

Smoothing is a method of noise reduction for image data. The measurement vector for each pixel is replaced by the average of the measurement vector for that pixel and the measurement vectors of the eight surrounding pixels. Consider the following example, as shown in Fig. 2. $x_{i,j}$, the measurement vector for pixel (i,j) is replaced by:

$$x'_{i,j} = (x_{i-1,j-1}+x_{i,j-1}+x_{i+1,j-1}+ \\ x_{i-1,j}+x_{i,j}+x_{i+1,j}+ \\ x_{i-1,j+1}+x_{i,j+1}+x_{i+1,j+1})/9$$

Thus, for each pixel, eight vector additions and one division of a vector by a constant is required. Consider the case where each measurement vector is 4-dimensional and the image is I-by-J pixels. Smoothing the image on a serial machine will require 8*I*J vector additions and I*J divisions, translating to 32*I*J additions and 4*I*J divisions.

If J is sufficiently large (> 2N+1) and a multiple of N, the image can be divided into N columns J/N pixels wide as shown in Fig. 3. This scheme is called *striping* and has been discussed in [3]. Each processor will process one stripe. In order to process all pixels in a given stripe, a processor will need to access one column of pixels from each bordering stripe. This means that at least two columns of data will have to be stored in shared memory. For example, with a 512-by-512 image and 32 PUs, PU 0 will process columns 0 to 15, while PU 1 will process columns 16 to 31, etc. Memory 0 will store column 0, memory 1 will store columns 1 through 14, memory 2 will store columns 15 and 16, etc. Note that memories 0 and 2 could contain more columns of data. In general, up to two rows of data must be stored in each shared memory. The rest of the image is stored in the local memory banks. The total processing time associated with an image is: 32*I*J/N additions and 4*I*J/N divisions. Thus, the theoretical maximum speedup by a factor of N is achieved.

If J is not a multiple of N, all PUs will process $\lfloor J/N \rfloor$ columns, then J mod N PUs will have to process one extra column of data. For simplicity, assume that columns cannot be subdivided. Thus, some processors will have to process a stripe $\lceil J/N \rceil$ columns wide, while other processors will have to process a stripe $\lfloor J/N \rfloor$ columns wide. If each column is J pixels wide, the total processing time associated with a given image will be:

$$32*I*(\lceil J/N \rceil) \quad additions$$

$$4*I*(\lceil J/N \rceil) \quad divisions$$

This represents an increase of at most 32*I additions and 4*I divisions over the ideal case. The efficiency of the above implementation can be represented by the ratio of the time required for an ideal speedup to the actual processing time [4]. This translates to:

$$\frac{J/N}{\lceil J/N \rceil}$$

The worst case efficiency is achieved when one processor is running while the remaining processors are idled. Mathematically, this is when the difference between J/N and $\lceil J/N \rceil$ is a maximum. For example, with N=1024 and an image with 4097 columns, this represents an efficiency of 80%, while for J=65537, this represents an efficiency of 98.4%. The larger the image, the closer the efficiency is to 100%.

Note that the efficiency is a function of the number of columns. Processing rows instead of columns will make the efficiency a function of the number of rows and may allow N PUs to operate more efficiently.

An alternative to the above method is to allow a column to be divided among PUs. Furthermore, some processors will process $\lceil I*J/N \rceil$ pixels, while others will process $\lfloor I*J/N \rfloor$. This scheme is shown in Fig. 4, and is called modified striping.

This time required to smooth an image using modified striping is:

$$32*\lceil I*J/N \rceil \quad additions$$

$$4*\lceil I*J/N \rceil \quad divisions$$

For the ideal speedup of N, the ceiling function would be absent, thus the ratio of the ideal speedup to the actual speedup becomes:

$$\frac{I*J/N}{\lceil I*J/N \rceil}$$

For N=1024, and an image of size 1025-by-4097, the efficiency is 99.99+%. This method, thus, leads to a higher overall utilization of the PUs. Further, for images greater than 2N-by-2N, the utilization is independent of the orientation of the image, i.e, whether the image is striped based on rows or columns.

If striping is done by columns, images smaller than 2N columns have not been considered, as they do not have enough columns to utilize the full machine. Each processor will have to store at least one column of data in each of its shared memory banks. This implies that there are at least two rows of data per processor. Multiplication of the two column minimum by the N processors yields 2N columns. If striping is done by rows, then the argument is similar. To process small images, $\lceil J/2 \rceil$ processors would have to be enabled, while the rest of the processors were disabled for the entire task.

IV. MAXIMUM LIKELIHOOD CLASSIFICATION

Maximum likelihood classification (MLC) [5] classifies each pixel independently of all others. Assume that the input data can be described by a Gaussian distribution function [5]. Thus, the probability that pixel (i,j) is in a given class $\omega_k \varepsilon \Omega = \{\omega_1, \omega_2, \cdots \omega_n\}$ is:

$$p(X_{ij} \mid \omega_k) = \frac{e^{-\frac{1}{2}(X_{ij}-M_k)^T \Sigma_k^{-1}(X_{ij}-M_k)}}{\sqrt{2\Pi^n \mid \Sigma_k \mid}}$$

where X_{ij} is the measurement vector for pixel (i,j), M_k is the mean vector for class k, Σ_k is the covariance matrix for class k. A pixel is assigned to a given class such that $p(X_{ij}\,|\,\omega_k)$ is maximized. It is possible to use a discriminant function [5]:

$$d(X_{ij}\,|\,\omega_k) = -\left[\ln|\Sigma_k| + (X_{ij}-m_k)^T\Sigma_k^{-1}(X_{ij}-m_k)\right]$$

Maximizing this last discriminant function for X_{ij} over Ω will yield the same result as maximizing $p(X_{ij}\,|\,\omega_k)$ over the same Ω. The discriminant function is considerably less complex to calculate than the probability, so discussion is based on the discriminant function.

The calculation of $-\ln|\Sigma_k|$ and Σ_k^{-1} is done once for each information class and is negligible when compared to the calculation of the discriminant function for each class for each pixel in a given image. Again assuming X_{ij} is 4-dimensional, $X_{ij}-m_k$ can be done in four additions per class per pixel. By utilizing the symmetry of Σ_k^{-1}, $(X_{ij}-m_k)\Sigma_k^{-1}(X_{ij}-m_k)$ can be performed in 20 multiplies and 9 additions for the four spectral band case. Thus, the calculation of the discriminant function will require 20 multiplies, 15 additions, and one sign change per pixel per class. Finally, for C class data, C-1 compares per pixel will be needed in addition to the calculation of the discriminant function. On an I-by-J image, classification of all I*J pixels will require 20*I*J*C multiplications, 15*I*J*C additions, and I*J*(C-1) compares for a standard serial processor.

Consider implementing the MLC on MuRSS. The CU will broadcast class dependent constants, such as Σ_k^{-1} and m_k as part of the SIMD program. Each pixel is classified independently, thus there is no need for any inter-PU communication. Using the modified striping scheme to divide the I-by-J image, N PUs will be able to perform an MLC

$$\frac{I*J\,/\,N}{\lceil I*J\,/\,N\rceil}\times N$$

times faster than a single PU. Further, since this operation requires no inter-PU data transfers, images as small as N pixels can be processed without disabling PUs for the entire operation.

V. CONTEXTUAL CLASSIFICATION

The "class" associated with a given pixel is not independent of the classes of adjacent 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. The image can be considered to be a two-dimensional random process incorporated into the classification strategy. This is the objective of "contextual classifiers" [6,7], in which a form of compound decision theory is employed through the use of a statistical characterization of context. Recent investigations have demonstrated the effectiveness of a contextual classifier that combines spatial and spectral information by exploiting the tendency of certain ground-cover classes to occur more frequently in some spatial contexts than in others [3,7,8,6].

The image data to be classified is 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 $X_{ij}\varepsilon R^n$ and the true class of the pixel $\omega_{ij}\varepsilon\Omega=\{\omega_1,\cdots,\omega_C\}$. The measurement vectors 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*. 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 Ω, a discriminant function g is calculated. The pixel is assigned to the class for which g is the greatest. Each value of g is computed as a weighted sum of the product of probabilities based on the pixels in the neighborhood. This is described below mathematically for pixel (i,j) being in class ω_k. (The description is followed by an example to clarify the notation used. Further details may be found in [3,7].)

$$g_k(X_{ij}) = \sum_{\substack{\underline{\omega}_{ij}\varepsilon\Omega^p,\\ \omega_{ij}=\omega_k}}\left[\left[\prod_{\gamma=1}^{p} f(X_\gamma\,|\,\omega_\gamma)\right]G^p(\underline{\omega}_{ij})\right]$$

where

$X_\gamma\varepsilon\underline{X}_{ij}$ is the measurement vector from the γth pixel in the p-array associated with pixel (i,j)

$\omega_\gamma\varepsilon\underline{\omega}_{ij}$ is the class of the γth pixel in the p-array associated with pixel (i,j)

$f(X_\gamma\,|\,\omega_\gamma)$ is the class-conditional density of X_γ given that the γth pixel is from class ω_γ

$G^p(\underline{\omega}_{ij}) = G(\omega_1,\omega_2,\cdots,\omega_p)$ is the a priori probability of observing the p-array $\omega_1,\omega_2,\cdots,\omega_p$

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*. The class-conditional density of pixel measurement vector X given that the pixel is from class k is:

$$f(X\,|\,k) = e^{\frac{-[\log|\Sigma_k| + (X-m_k)^T\Sigma_k^{-1}(X-m_k)]}{2}}$$

where the measurement vector for each pixel is of size four, Σ_k^{-1} is the inverse of the 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, "log" is the natural logarithm, and $|\Sigma_k|$ is the determinant of the covariance matrix. This is the same function as used for the MLC [5].

Consider, as an example, the horizontally linear neighborhood shown in Fig. 5, and assume there are two possible classes: $\Omega\varepsilon\{a,b\}$. Then the discriminant function for class b is explicitly (pixel (i,j) is the middle pixel):

$$g_b(X_{ij}) = f(X_1|a)f(X_2|b)f(X_3|a)G(a,b,a)$$
$$+f(X_1|a)f(X_2|b)f(X_3|b)G(a,b,b)$$
$$+f(X_1|b)f(X_2|b)f(X_3|a)G(b,b,a)$$
$$+f(X_1|b)f(X_2|b)f(X_3|b)G(b,b,b)$$

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

Consider the case where there is a nonlinear three-by-three context array (neighborhood), as shown in Fig. 6. Here, for each g, there are many more products to compute, each with more factors than the one-by-three case. In general, for each g, there are C^{p-1} product terms, each term having p+1 factors, where C is the number of classes and p is the neighborhood size. All of the calculations are done using floating point arithmetic. It is the parallel implementation of contextual classifiers that is under consideration.

The algorithm shown in Fig. 7 is a uniprocessor implementation of the size three contextual classifier. Let "hold(m,k)" be a two-dimensional array of size three-by-C, i.e., $0 \le m \le 2$ and $1 \le k \le C$. "hold(cr,k)" is a vector of length C containing the class-conditional density values ("compf"s) for the pixel (i,j) ("cr" is an abbreviation for center). "hold(lt,k)" and "hold(rt,k)" are the analogous vectors for the pixel (i,j-1) (the left neighbor) and pixel (i,j+1) (the right neighbor), respectively. By using this array to save the class-conditional densities, each density (for a given pixel and class) is calculated only once [9].

The complexity of the algorithm is proportional to I*J*C^3 assignments, multiplications, and additions, and I*J*C "compf" calculations. Typically, $10 \le C \le 60$ for the analysis of LANDSAT data.

The algorithm can be extended for a non-linear contextual classifier with a neighborhood of size nine (as shown in Fig. 6) [10]. The complexity of the algorithm would have growth proportional to $I*J*C^9$ assignments, multiplications, and additions. The number of "compf" calculations would still be I*J*C. In this case, "hold" would be a ((2*J)+3)-by-C array (assuming the neighborhood window moves along rows). The (2*J)+3 pixels whose "compf" values are stored in "hold" are chosen to make it unnecessary to perform redundant "compf" calculations. In general, when classifying pixel (i,j), "hold" has the "compf" values for pixels j-1 to J-1 of row i-1, pixels 0 to J-1 (all) of row i, and pixels 0 to j+1 of row i+1. After the classification of pixel (i,j), the values for (i+1,j+2) are stored and the values for (i-1,j-1) are deleted. When the pixels on a new row are to be classified, call it i', then the values for pixels (i'-2,J-3), (i'-2,J-2), and (i'-2,J-1) are removed and the values for (i'+1,0), (i'+1,1), and (i'+1,2) are stored. (This assumes row i' is classified after i'-1). Given this, the rest of transforming the algorithm for the size nine square neighborhood case is straightforward.

For the three-by-three window, data allocation and timing analysis is analogous to that for smoothing. The main difference is that for smoothing, only the raw pixel data is shared. For the contextual classifier, the "compf" values of the subimage edge pixels are shared instead. The parallel processor version of the one-by-three horizontally linear window is similar. Other sizes and shapes of windows are handled analogously.

VI. IMAGE CORRELATION ON A PARALLEL MACHINE

Image correlation, as described in [11], is used to measure the degree of similarity between a match image and an equal sized area of an input image. Typical images can be at least 4096-by-4096 pixels, with match areas on the order of 64-by-64 pixels. For the purposes of this paper, images on the order of 65536-by-65536 pixels will be considered.

Let the symbols x and y denote single elements of arrays X and Y, where X is the match image and Y is the area of the input image under consideration (same dimensions as X). Let M be the total number of elements in the match area X. Define:

$$S_{XX} = (1/M)(\sum x^2 - (\sum x)^2)$$
$$S_{XY} = (1/M)(\sum xy - \sum x \sum y)$$
$$S_{YY} = (1/M)(\sum y^2 - (\sum y)^2)$$
$$R_{XY} = S_{XY} / \sqrt{S_{XX} S_{YY}}$$

S_{XY} is the covariance of the match area with a portion of the input area. Large positive values for S_{XY} indicate similarity between the match image and the input image, while large negative values for S_{XY} indicate similarity between the negative of the match image and the input image. Values near zero indicate little similarity between the two images. R_{XY} is the linear correlation coefficient of the statistics. Simplistically R_{XY} is a normalized version of S_{XY} in which $R_{XY} = 1$ indicates an identical match, $R_{XY} = -1$ indicates an identical match with the negative of the input area, and $R_{XY} = 0$ indicates no correlation between the match area and the input image. A correlation value will be computed for each position in which the match image can fit into the R row by C column input image.

The calculation of R_{XY} is dominated by the time to compute $\sum xy$, $\sum y$, and $\sum y^2$. $\sum x$ and $\sum x^2$ do not change from input window to input window, and can thus be pre-computed. For a match template with r rows and c columns, each $\sum xy$ and $\sum y^2$ requires r*c multiplications and rc-1 additions. $\sum y$ requires rc-1 additions. These operations have to be done for each position of the match template in the input image. Special methods of computing $\sum y^2$ and $\sum y$ can decrease the time requirements of this algorithm. Consider the following algorithm for computing the sum of the pixel values ($\sum y$'s) in each match template.

Assume that for input image Y the position of the match area is defined by the coordinates of the upper left hand corner of the match area. Define a vector "colsum" [11] of length C as:

$$colsum(j) = \sum_{i=k}^{k+r-1} Y(i,j)$$

where k is the row coordinate of the current portion of the match area and $0 \le j < C$. Let "SUM" be an R-r+1 by C-c+1 array, where SUM_{ij} is the sum of the pixels of the input image for the match area position (i,j), $0 \le i \le R-r+1$, $0 \le j \le C-c+1$.

Initially, colsum is calculated for all C columns of row 0. SUM(0,0) is formed by summing colsum(j) ($0 \le j \le c-1$). This requires r*c multiplications and (r*c)-1 additions. SUM(0,1) is formed by subtracting colsum(0) from SUM(0,0) and adding colsum(c) to the result. In general:

$$SUM(0,j) = SUM(0,j-1) - colsum(j-1) + colsum(j+c-1)$$

After the processing of a given row is complete, colsum(j) is updated for the next row by subtracting $Y(ij)$ from the old colsum(j) and adding $Y(i+r-1,j)$ to the result. This changes the complexity for the calculation of the $\sum y$'s to: 3c-1 additions/subtractions per template position for the column 0 entries of all other rows, and 4 additions/subtractions per template position for all other template positions.

For a typical 64-by-64 match image, straight forward computation of $\sum y$ requires 4095 additions per match template position on the input image. This is the same number of operations required per match template position in row 0 of the input image. For template positions in column 0 of the other rows, 191 additions are required. Computation of $\sum y^2$'s is similar to the computation of the $\sum y$'s.

Consider the application of MuRSS to this task. Each PU will apply the serial algorithm to its assigned pixels. Pixels will be assigned to PUs based on the vertical striping scheme. If a column of pixels lies in memory associated with bus 0 or bus 1 of PUi, then PUi is responsible for the computation of the colsum and the analogous y^2 entries associated with that column. If the pixel in the upper left hand corner of a window lies in memory associated with bus 0 or bus 1 of PUi, then PUi is responsible for the computation of that window. When PUi is performing computations on its rightmost c-1 columns, it uses the colsum values stored in its bus 2 memory by the previous computations of PUi+1 (recall that PUi+1s bus 0 memory is PUi's bus 2 memory). Thus, at least c-1 colsum values and the corresponding y values must be stored in memory associated with each bus 0.

For an R-by-C image and N PUs, the vertical striping scheme will assign each PU a subimage either R-by-$\lceil C/N \rceil$ or R-by-$\lfloor C/N \rfloor$. Thus, the total time required for the calculation of the $\sum xy$'s is (R-r+1)*($\lceil C/N \rceil$-c+1)*((r*c)-1) additions, and (R-r+1)*($\lceil C/N \rceil$-c+1)*r*c multiplications. The total time associated with the calculation of the $\sum y$'s is $\left[(R-r)*((3*c)-1)\right] + \left[(\lceil C/N \rceil - c+1)*((r*c)-1)\right] + \left[(R-r)*(\lceil C/N \rceil - c)*4\right]$ additions. The time required to calculate the $\sum y^2$'s is similar to the time associated with the calculation of the $\sum y$'s.

If $C < N*(c-1)$, then c-1 columns of data cannot be associated with each bus 0, thus the PUs cannot all be enabled. If $R \ge N*(r-1)$, the stripes can be horizontal instead of vertical. In this case, r and c are swapped, as well as R and C.

VII. CONCLUSIONS

MuRSS, an SIMD architecture with as many as 1024 processors, was presented. It was shown that N processors in the SIMD mode of operation could perform various image processing tasks almost N times faster than one processor of the same type. Four tasks considered were smoothing, maximum likelihood classification, contextual classification, and image correlation. The application of the MuRSS to the tasks considered was discussed.

Through the use of the MuRSS SIMD architecture, computationally demanding remote sensing processes can be implemented efficiently. This will not only reduce the computation time required to perform remote sensing tasks, but will also allow the investigation of techniques which may otherwise be considered infeasible.

REFERENCES

[1] M.J. Flynn, "Very high speed computing systems," *Proc. IEEE*, Vol. 54, pp. 1901-1090, Dec. 1966.

[2] *MC68000, 16-Bit Microprocessor User's Manual*, Motorola Inc., Austin, Texas, 1980.

[3] P.H. Swain, H.J. Siegel, and B.W. Smith, "Contextual classification of multispectral remote sensing data using a multiprocessor system," *IEEE Trans. Geoscience and Remote Sensing*, Vol. GE-18, pp. 197-203, Apr. 1980.

[4] L.J. Siegel, H.J. Siegel, and P.H. Swain, "Performance measures for evaluating algorithms for SIMD machines," *IEEE Trans. Software Engineering*, scheduled to appear July 1982.

[5] P.H. Swain and S. Davis, editors, *Remote Sensing: The Quantitative Approach*, McGraw-Hill Inc., New York, NY, 1978.

[6] J.R. Welch and K.G. Salter, "A context algorithm for pattern recognition and image interpretation," *IEEE Trans. Systems, Man, and Cybernetics*, Vol. SMC-1, pp. 24-30, Jan. 1971.

[7] P.H. Swain, S.B. Vardeman, and J.C. Tilton, "Contextual classification of multispectral image data", *Pattern Recognition*, Vol. 13, pp. 429-441, Mar. 1981.

[8] J.C. Tilton, P.H. Swain, and S.B. Vardeman, "Contextual classification of multispectral image data: an unbiased estimator for context distribution," *1981 Symposium on Machine Processing of Remotely Sensed Data*, Purdue University, pp. 304-313, June 1981

[9] H.J. Siegel, P.H. Swain, and B.W. Smith, "Parallel processing implementations of a contextual classifier for multispectral remote sensing data", *Proc. 1980 Machine Processing of Remotely Sensed Data Symposium* (IEEE Catalog No. 80 CH 1430-8 MPRSD), pp. 19-29, June 1980.

[10] B.W. Smith, H.J. Siegel, and P.H. Swain," Contextual classification on a CDC Flexible Processor system", *Proc. of the 1981 machine Processing of Remotely Sensed Data Symposium* (IEEE Catalog No. 81 CH 1430-8 MPRSD), pp. 283-288, June 1981.

[11] L.J. Siegel, H.J. Siegel, and A.E. Feather, "Parallel processing approaches to image correlation," *IEEE Trans. Computers, Vol. C-34*, pp. 208-217, Mar. 1982.

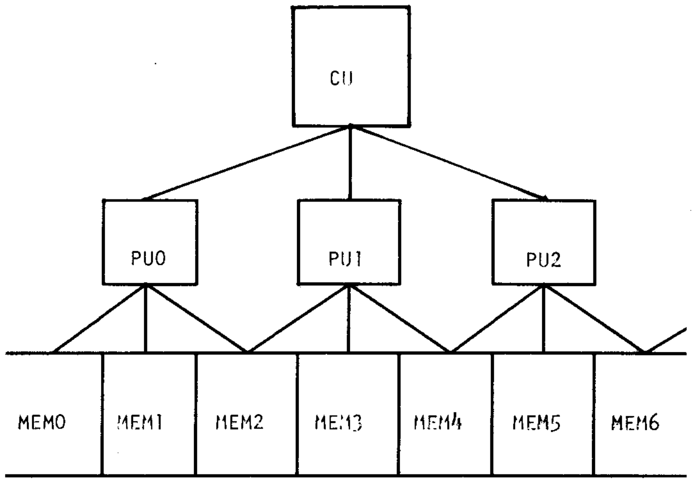

Fig. 1 MURSS processor/memory arrangement

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

Fig. 2 Pixels used in the calulation of pixel (i,j) when smoothing an image.

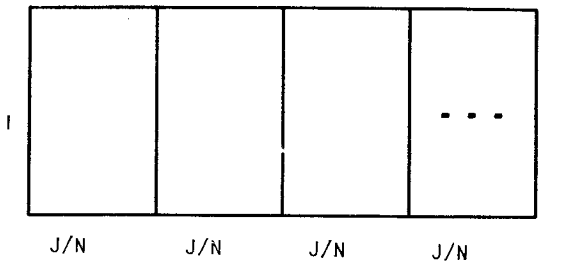

Fig. 3 Striping scheme of dividing an I-by-J image among N PUs.

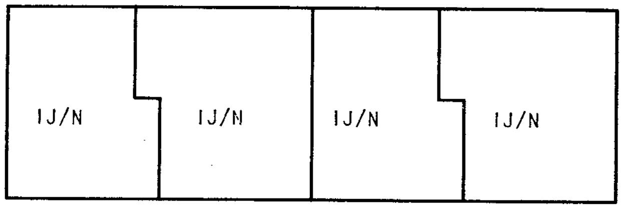

Fig. 4 Modified striping scheme of dividing an I-by-J image among N PUs.

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

Fig. 5 Linear neighborhood of size three.

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

Fig. 6 Non-linear neighborhood of size nine.

Main Loop

```
for i = 0 to I-1 do /* row index */
   for k = 1 to C do /* for each class */
      for m = 0 to 2 do hold(m,k) = compf(i,m,k) /*cols.0-2*/
   lt = 0 /* hold(lt,k) is left neighbor */
   cr = 1 /* hold(cr,k) is pixel being classified */
   rt = 2 /* hold(rt,k) is right neighbor */
   for j = 1 to J-2 do /* column index */
      value = -1; class = -1 /* max "g" and class */
      for k = 1 to C do /* for each class */
         current = g(lt,cr,rt,k)
         if current > value /* compare with max */
            then value = current; class = k
      print pixel (i,j) is classified as "class"
      if j ≠ J-2 then /* update hold pointers */
         tp = lt; lt = cr; cr = rt; rt = tp
         for k = 1 to C do /* compf's for next col */
            hold(rt,k) = compf(i,j+2,k)
```

Discriminant Function Calculation

```
function g(lt,cr,rt,k)  /* for pixel cr, class k  */
sum = 0  /* initialize sum, used to accumulate g  */
for r = 1 to C do  /* all classes for pixel (i,j-1)  */
   for q = 1 to C do  /* all classes for pixel (i,j+1)  */
      if G(r,k,q) ≠ 0  /* do not multiply if G = 0  */
         then sum = hold(lt,r) * hold(cr,k)
                   * hold(rt,q) * G(r,k,q) + sum
return (sum) /* sum contains value of g(lt,cr,rt,k)  */
```

Class-Conditional Density Calculation

```
function compf(a,b,k)  /* for pixel (a,b), class k  */
x = A(a,b)  /* x is the pixel (a,b) measurement vector  */
expo = $-[\log|\Sigma_k| + (x-m_k)^T\Sigma_k^{-1}(x-m_k)]/2$
return ($e^{expo}$)  /* return value of f(A(a,b)|k)  */
```

Fig. 7 Uniprocessor version of contextual classification algorithm.