1-1-1981

Contextual Classification on a CDC Flexible Processor System

B. W. Smith

H. J. Siegel

P. H. Swain

Reprinted from

Seventh International Symposium

Machine Processing of

Remotely Sensed Data

with special emphasis on

Range, Forest and Wetlands Assessment

June 23 - 26, 1981

Proceedings

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

CONTEXTUAL CLASSIFICATION ON A CDC FLEXIBLE PROCESSOR SYSTEM

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

Purdue University

ABSTRACT

Contextual classifiers are being developed to exploit spatial/spectral content of a pixel to achieve higher classification accuracy. Contextual classification requires large amounts of computation, so special hardware is of value. One parallel processing system designed for image processing is the CDC Flexible Processor Array, the basis for the discussion in this paper. A simulator for the CDC Flexible Processor Array has been developed for program testing, debugging and timing. The simulated timings are presented in this paper. For comparison, the same algorithms have been run on a PDP-11/70. These timings are analyzed and discussed for context neighborhoods of size three and nine.

I. INTRODUCTION

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. The image can be considered to be a two-dimensional random process and the characteristics of this process incorporated into the classification strategy. This is the objective of "contextual classifiers" [4,5] in which a form of compound decision theory is employed to improve scene classification through use of a statistical characterization of context. These classifiers are an extension of an idea by Welch and Salter[6].

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 CDC Flexible Processor system is a commercially available multiprocessor system which has been recommended for use in remote sensing [1,2,3].

Section II briefly describes the contextual classifier and gives an algorithm for performing it. Section III presents uniprocessor classification algorithms. Section IV presents the Flexible Processor algorithm, a potential Flexible Processor Array organization, and timings for the contextual classifiers.

II. CONTEXTUAL CLASSIFIERS

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

---

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

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 Ω, a discriminant function g is calculated. The pixel is assigned to the class for which g is 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 [4,5].)

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

where

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

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

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

$G^p(\underline{\theta}_{ij})$ = G $(\theta_1, \theta_2, \ldots, \theta_p)$ 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. The class-conditional density of pixel measurement vector x given that the pixel is from class k is:

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

where the measurement vector for each pixel is of size n=4, Σ_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. This is the same function as used for maximum likelihood classification [7].

Consider as an example the horizontally linear neighborhood[4] shown in Fig. 1, and assume there are two possible classes: $\Omega = \{a,b\}$. Then the discriminant function for class b is explicitly:

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

(Recall $G(\theta_1,\theta_2,\theta_3)$ is the a priori probability of observing the specific neighborhood configuration $(\theta_1,\theta_2,\theta_3)$.) 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 non-linear three-by-three context array (neighborhood), shown in Fig. 2. 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 data. It is the parallel implementation of contextual classifiers that is the subject of this paper.

III. UNIPROCESSOR CONTEXTUAL CLASSIFIERS

The algorithm, shown in Fig. 3, implements the contextual classifier. Let "hold(m,k)" be a two-dimensional array of size three-by-C, i.e., $0 \le m \le 2$ and $1 \le k \le C$. For m=cr, hold(cr,$\overline{k}$) 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). For example, hold(cr,1) is the class-conditional density for pixel (i,j) and class 1. hold(lt,k) and hold(rt,k) are the analogous vectors for the pixel (i,j-1) (the left ("lt") neighbor) and pixel (i,j+1) (the right ("rt") neighbor), respectively. By using this array to save the class-conditional densities, each density (for a given pixel and class) is calculated only once.

To reduce the number of floating point operations in "g'(lt,cr,rt,k)" for the algorithm, the sum is updated only if "G(r,k,q)" is non-zero. In the Landsat data used in the testing described in [5], the percentage of a priori probabilities (G^9s) that were non-zero was about 1% (based on a size nine neighborhood and 14 classes). Thus, most of the G^9s that are stored are zeroes. The memory requirements of the classifier can be reduced greatly if the zero values are ignored and only the non-zero values stored. Assume that each non-zero G value is a floating point number requiring 32 bits. In memory, alternate each non-zero G value with a 16-bit word that specifies the three classes associated with that class, e.g., if G(3,3,2) is non-zero, the word preceding it is a representation (concatenation) of 3, 3, and 2. This would allow $\lfloor 16/3 \rfloor$ = 5 bits per pixel for specifying the class, i.e., up to 32 classes. This data compaction method would be useful whenever more than one-third of the G's are zero. Variations on this method may be employed for larger neighborhhoods and greater numbers of classes.

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

The given algorithm can be extended for a non-linear contextual classifier with a neighborhood of size nine (as shown in Fig. 2). 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.

For the size nine square neighborhood case, "hold" would be a (2*J+3)-by-C array (assuming the neighborhood window moves along rows). The "C" term is for holding the C "compf" values that are calculated for a pixel. 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 pixel (i-1,j-1) are removed from "hold" and values for (i+1,j+2) are added. 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 added. (This assumes row i is classified after i -1.) If J < I, then moving the neighborhood window along columns would save space, since "hold" would then be of size (2*I+3)C. Given this, the rest of the transformation to the size nine square neighborhood case is straightforward.

IV. CONTEXTUAL CLASSIFIERS ON AN FP ARRAY

The Control Data Corporation Flexible Processor system is a multiprocessor system which has been recommended for use in remote sensing. The basic components of a Flexible Processor (FP) are shown in Fig. 4. There can be up to 16 FPs linked together, providing much parallelism at the processor level. The FPs can communicate among themselves through a high-speed ring or shared bulk memory. The clock cycle time of each FP is 125 nsec. Since 16 FPs can be connected in a parallel or pipelined fashion, the effective throughput can be drastically increased.

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, execute 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 of over 8000 16-bit words.
Up to 16 banks of 250 nsec. bulk memory (each bank holds 64k words).

In order to debug, verify and time FP algorithms, a simulator was developed for an array of up to 16 FPs. An assembler for the micro-assembly level language was also developed. Both are designed to operate under the UNIX operating system. They are described in [8]. Their use in programming and executing a maximum likelihood classifier is discussed in [4]. The FP and the array are covered in depth in [1,2,8].

Consider using an FP system to implement the contextual classifier based on a horizontally linear neighborhood of size three (Fig. 1). Divide the A-by-B image into subimages of B/N rows A pixels long, as shown in Fig. 5. 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 on its own subimage. No interaction between FPs is needed, i.e., each FP can process its subimage independently.

An FP micro-assembly language version of the algorithm stated in Fig. 3 was coded and debugged. Because each FP is microprogrammable, determining program correctness and analyzing the execution time is done through the use of the micro-assembler and simulator. Execution times per pixel vary because all floating point operations are done in the software. The classification time associated with the first pixel on a line is different from the classification of the rest of the pixels on the same line, since data must be calculated for each of the pixels in the window. In all remaining windows, data must be calculated for only one pixel.

The pixel measurement vectors, covariance matrices, logarithms of the determinants of the covariance matrices, a priori probabilities, and hold array are all stored in the Large File (see Fig. 4). In this way, each FP has all the information it needs for performing the classification on its subimage.

If the number of non-zero a priori probabilities is small (less than 50%), and the contextual information associated with each G^P can be stored in the space of one floating point number (32 bits), then any algorithm that stores all a priori probabilities will waste space. This was the case in the Landsat data used for this experiment. To conserve memory space, the contextual information was stored in the floating point location immediately preceding the G^P that it was to identify.

For the purpose of testing the FP contextual classifier program, 30 rows of 16 pixel measurement vectors were classified. Each measurement vector consisted of four 32-bit floating point representations of 8-bit integers. All data were stored in the Large File. The data set consisted of a four-class subset of the data used in [5]. To provide a basis for comparison, a similar contextual classifier was run on a PDP-11/70 over the same test data. It was found that lack of exponent range in the 11/70 floating point hardware required extra handling. FP floating point algorithms are implemented in the software, so a 14-bit exponent was used to overcome this problem. Twenty non-zero GPs were chosen for the benchmark tests. Running under the above constraints, the single FP classifier took .035 sec./pixel, while the PDP-11/70 required .050 sec./pixel. If the image data are too large to fit in the register files, bulk memory can be used, adding, at most, 1 microsec./pixel to the classification time, assuming one 16-bit bus between an FP and its associated banks of bulk memory.

Using .05 sec. per pixel as the PDP processing time, and .035 sec. per pixel as the single FP processing time, a 16 FP configuration, where each processor had its own bulk memory, would perform contextual classifications at a rate of 457 pixels per sec., as opposed to 20 pixels per sec. for a single PDP-11/70. There are, of course, cost differences between these two systems; however, the purpose here is to show the gains made possible by a multiprocessor system.

Consider horizontally linear neighborhoods, in general, such as those shown in Fig. 6. When using N FPs together to process an image, each FP handles 1/Nth 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 [4].) Vertically linear and diagonally linear neighborhoods (Fig. 7) can be processed in a manner similar to that for horizontally linear neighborhoods[4].

Consider non-linear neighborhoods, i.e., neighborhoods which do not fit into one of the linear classes. For example, all of the neighborhoods in Fig. 8 are non-linear. 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 non-linear neighborhood can be performed without sharing data among FPs.

The speed at which the contextual classifier runs depends on the floating point algorithms which are implemented in the software. The floating point routines require variable amounts of time based on the number of shifts required to normalize the data. This can cause a bottleneck in the processing if one FP is required to wait for another. Synchronization can require large amounts of time if the full 16 processor array is used.

Consider the non-linear neighborhood as shown in Fig. 2. Each box represents one pixel, while the numbers in each box refer to the numbering used to distinguish the various pixels. The use of a non-linear context neighborhood implies that certain data must be shared among the FPs. For example, assume that the data for pixels 1, 2, 4, 5, 7 and 8 are stored in FP K, and that the data for pixels 3, 6 and 9 are stored in FP K+1. FP K will need to communicate with FP K+1 to obtain the data necessary to classify pixel 5.

An FP is capable of addressing up to three channels of 16-by-128K bytes of bulk memory each[1,2]. The sharing of bulk memory is a scheme that can be used for shared data. One possible implementation is shown in Fig. 9. Assume each FP will classify the pixels in B/N rows (Fig. 5). If border areas are stored in the joint memory banks, a processor will begin processing in banks of bus 1. Processing will continue through half the banks in bus 1 to bank 0 on bus 2. After all the data in the banks on data bus 2 have been processed, processing will continue to the banks on bus 3.

Allowing 25% of FP i's data to be stored in the shared banks on bus 1, 50% of the data to be stored in the local banks on bus 2, and 25% of of the data to be stored in the shared banks on bus 3, no contention will occur. Consider that for processor i to "catch up" with processor i+1, processor i will have to process more than 75% of its data in the time that it takes processor i+1 to process 25% of its data. Contention is not a problem.

An FP will be allowed to address only half of its memory banks at one time. This is done to facilitate double buffering. The other half will be accessible by the host. This allows, for example, the FP to be classifying the current image while the host unloads and stores the results of the previous classification and then loads the next image to be processed.

The eight-nearest-neighbor contextual classifier is similar to the previously discussed linear case. Differences arise in the calculation of the discriminant function, the method of updating the data for a given window, and the method of data storage.

The calculation of the discriminant function for a given class requires that the class-conditional densities must be used from the eight surrounding pixels, instead of the class-conditional densities for the pixels on just the left and the right. From probability and measure theory, it can be seen that the increase in calculations is exponential instead of linear. Further, the potential number of stored a priori probabilities grows at the same rate. Within the space limitation of the Large File, either the number of classes must be kept small or only the non-zero probabilities must be stored. In some cases, it may be necessary to store and use only G values above a certain threshold. The final difference between the linear neighborhood and the non-linear neighborhood is that when the window is moved, the data in the linear case are shifted from pixel 3 to pixel 2 and from pixel 2 to pixel 1, while in the non-linear case, the data must be moved from pixel 3 to pixel 2, pixel 2 to pixel 1, pixel 6 to pixel 5, pixel 5 to pixel 4, pixel 9 to pixel 8, pixel 8 to pixel 7.

Timings run with Landsat data from [5] show that, on the average, the FP implementation of the four class, size nine square neighborhood contextual classifier requires .137 sec./pixel. A PDP-11/70 implementation of the same algorithm requires .154 sec./pixel. Tests for the 11/70 were run with 50 non-zero GP and 4 spectral classes on 52 lines of 16 pixels. A 30-line by 16-pixel subset of the above image was used to derive the FP timings for a 52-line image. Pixels on the top and bottom line of an image are not classified, and thus do not appear in the number of classified pixels. As a result, for the first and last rows of an image, the classifier must calculate the compf values for these pixels without ever classifying them. Only the non-zero G^P is stored, so only the non-zero G^P affects computation time. Based on the above timings, a 16-FP array can classify 116 pixels/sec., while a PDP-11/70 can classify 6 pixels/sec.

In summary, the organization of the FP system given above will allow contention-free sharing of data. This means that N FPs will be able to operate N times faster than one FP. Furthermore, the double-buffering of the bulk memories will allow the loading of images to be processed and storage of results to be overlapped with the classification operation of the FPs.

V. CONCLUSIONS

A potential hardware organization for the Flexible Processor Array was presented. Timings for the contextual classifier on a PDP-11/70 and on an FP were given. With the suggested hardware configuration, N FPs could perform contextual classification N times faster than a single FP system. This demonstrates the usefulness of parallelism for executing computationally intensive remote sensing algorithms.

Acknowledgements. The authors acknowledge many useful discussions with J. C. Tilton, who supplied the PDP execution time data.

REFERENCES

1. Control Data Corp., Cyber-Ikon Image Processing System Concepts. Digital Systems Division, Control Data Corp., Minneapolis, MN, Jan. 1977.

2. Control Data Corp., Cyber-Ikon Flexible Processor Programming Textbook.

Digital Systems Division, Control Data Corp., Minneapolis, MN, Nov. 1977.

3. J.L. Kast, P.H. Swain, T.L. Phillips, The Feasibility of Using a Cyber-Ikon System as the Nucleus of an Experimental Agricultural Center. LARS Contr. Report 021678, Laboratory for Applications of Remote Sensing (LARS), Purdue Univ., W. Lafayette, IN 47907, Feb.. 1978.

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

5. P.H. Swain, S.B. Vardeman, J.C. Tilton, Contextual Classification of Multispectral Image Data. LARS Contr. Report 011080, Laboratory for Applications of Remote Sensing, Purdue Univ., W. Lafayette, IN 47907, 34 pp., Jan. 1980.

6. J.R. Welch, 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.

7. P.H. Swain, S.M. Davis, Remote Sensing: The Quantitative Approach, McGraw-Hill, Inc., New York 1978.

8. B.W. Smith, H.J. Siegel, P.H. Swain, A Multiprocessor Implementation of a Contextual Image Processing Algorithm. AgRISTARS Report SR-PO-00474 (also LARS Technical Report 070180), Laboratory for Applications of Remote Sensing (LARS), Purdue Univ., W. Lafayette, IN 47907, July 1980.

1 (i,j-1)	2 (i,j)	3 (i,j+1)

Figure 1. A p=3 context array (neighborhood). The number above the row and column indices is the pixel number for that position in the neighborhood.

1 (i-1,j-1)	2 (i-1,j)	3 (i-1,j+1)
4 (i,j-1)	5 (i,j)	6 (i,j+1)
7 (i+1,j-1)	8 (i+1,j)	9 (i+1,j+1)

Figure 2. Three-by-three neighborhood for classifying pixel (i,j). The number above the row and column indices is the pixel number for that position in the neighborhood.

Main Loop for Algorithm

```
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 */
            /* "g" for pixel i,j class k */
            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 /* not last column */
            /* 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)
```

Figure 3. Implementation of a contextual classifier.

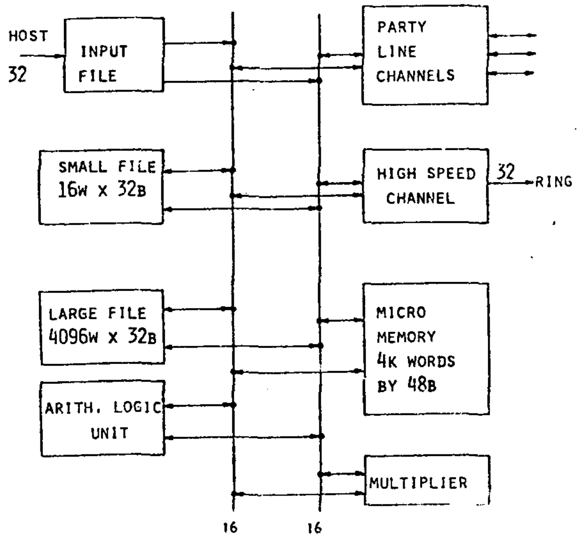

Figure 4. Data Paths in a Flexible Processor.

Discriminant Function Calculation for Algorithm

```
function g'(lt,cr,rt,k)
        /* discriminant for pixel "cr" (whose neighbors
           are "lt" and "rt") and class k */
sum = 0  /* initialize sum, used to accumulate g'(lt,cr,rt,k) */
for r = 1 to C do  /* all possible classes for pixel (i,j-1) */
    begin
    for q = 1 to C do  /* all possible classes for pixel
                          (i,j+1) */
        begin
        if G(r,k,q) ≠ 0  /* do not do multiplications if G = 0 */
            then
            sum = hold(lt,r) * hold(cr,k) * hold(rt,q)
                  * G(r,k,q) + sum
        end
    end
```

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 3 (cont.). Discriminant function and class-conditional density routines.

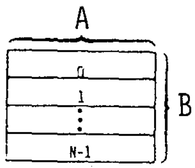

Figure 5. An A-by-B image divided among N Flexible Processors.

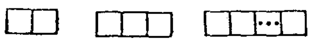

Figure 6. Horizontally linear neighborhoods.

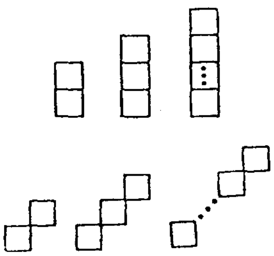

Figure 7. Vertically and diagonally linear neighborhoods.

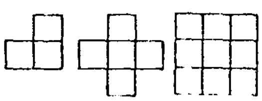

Figure 8. Non-linear neighborhoods.

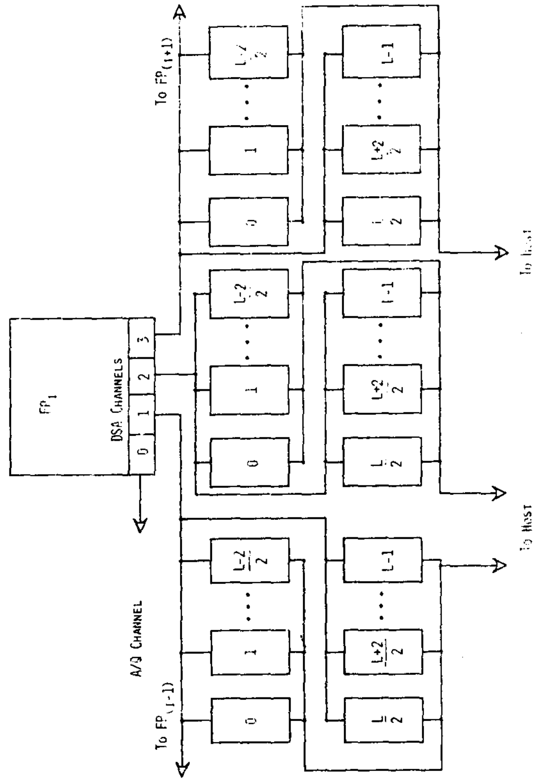

Figure 9. Potential memory organization for striping scheme.