1-1-1981

Contextual Classification on a CDC Flexible Processor System

B.W. Smith
H.J. Siegel
P.H. Swain

Follow this and additional works at: http://docs.lib.purdue.edu/lars_symp

http://docs.lib.purdue.edu/lars_symp/435

This document has been made available through Purdue e-Pubs, a service of the Purdue University Libraries. Please contact epubs@purdue.edu for additional information.
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 $X_{ij} \in \mathbb{R}$ 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 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 \( w_k \). (The description is followed by an example to clarify the notation used. Further details may be found in [4,5].)

\[
g_k(X_{ij}) = \sum_{\theta_{ij} \in \Omega} \left[ \prod_{l=1}^{p} f(X_{ij} | \theta_{ij}) \ G^P(\theta_{ij}) \right]
\]

where
\[X_{ij}\] is the measurement vector from the \( l \)th pixel in the p-array (for pixel \((i,j)\))
\[\theta_{ij}\] is the class of the \( l \)th pixel in the p-array (for pixel \((i,j)\))
\[f(X_{ij} | \theta_{ij})\] is the class-conditional density of \( X_{ij} \) given that the \( l \)th pixel is from class \( \theta_{ij} \)
\[G^P(\theta_{ij}) = G(\theta_1, \theta_2, ..., \theta_p)\] is the a priori probability of observing the p-array \( \theta_1, \theta_2, ..., \theta_p \).

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:

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

where the measurement vector for each pixel is of size \( n \times 4 \), \( \Sigma_k^{-1} \) 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. 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:

\[
g_b(X_{ij}) = f(X_{1}|a) f(X_{2}|b) f(X_{3}|a) G(a,b,a) \\
+ f(X_{1}|a) f(X_{2}|b) f(X_{3}|b) G(a,b,b) \\
+ f(X_{1}|b) f(X_{2}|b) f(X_{3}|a) G(b,b,a) \\
+ f(X_{1}|b) f(X_{2}|b) f(X_{3}|b) G(b,b,b)
\]

(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 \( C^2-1 \) 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-four, i.e., \( 0 \leq m < 2 \) and \( 1 \leq k < C \). For \( m=c \), hold(left, k) is a vector of length \( C \) containing the class-conditional density values ("compf"s) for the pixel \((i,j)\) ("lt" is an abbreviation for center). For example, hold(left, 1) is the class-conditional density for pixel \((i,j)\) and class 1. hold(left, k) and hold(right, 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 \( G(q) \) that were non-zero was about 1% (based on a size nine neighborhood and 14 classes). Thus, most of the G's that are stored are zeroes. The memory require-
ments 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 $[16/3] = 5$ 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 neighborhoods and greater numbers of classes.

The complexity of this algorithm is proportional to $I^2J^2C^2$ assignments, multiplications, and additions, and $I^2J^2C^2$ "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 be proportional to $I^2J^2C^2$ assignments, multiplications, and additions. The number of "compf" calculations would still be $I^2J^2C^2$.

For the size nine square neighborhood case, "hold" would be a $(2J+3)\times C$ array (assuming the window moves along rows). The "C" term is for holding the C "compf" values that are calculated for a pixel. The $2J+3$ 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 the $2J+3$ pixels to J-l of row i-l, pixels 0 to J-l (all of) row i, and pixels 0 to J+l of row i+l. 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 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 but would then be of size $(2I+3)C$. 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 multi-processing 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. 1. An assembler for the micro-assembly language level 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 $I$'s 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-assembly 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. The floating point algorithms which are implemented in the software. The floating point routines are a bottleneck in the software. The floating point routines required for one FP require extra handling.

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 set. The floating point algorithm required an extra handling. Each floating point operation is done in the software. The execution time for the algorithm stated in Fig. 3 was tested by running under the above constraints.

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 multi-processor system.

Consider horizontally linear neighborhoods, in general, such as those shown in Fig. 6. When using N FFs 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 when using N FFs together to process data among FFs.

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 FFs. For example, assume that the data for pixels 1, 2, 4, 5, 6 and 9 are stored in FF K, and that the data for pixels 3, 6 and 9 are stored in FF K+1. FF K will need to communicate with FF 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 each[1,2].
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 in case 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 calculation cost 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 GP 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 non-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


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.

<table>
<thead>
<tr>
<th>1 (i-1,j-1)</th>
<th>2 (i-1,j)</th>
<th>3 (i-1,j+1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 (i,j-1)</td>
<td>5 (i,j)</td>
<td>6 (i,j+1)</td>
</tr>
<tr>
<td>7 (i+1,j-1)</td>
<td>8 (i+1,j)</td>
<td>9 (i+1,j+1)</td>
</tr>
</tbody>
</table>

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 */
        It = 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) */
                G(r,k,q) = 1                   /* do not do multiplications if G,. = 0 */
                then sum = hold(lt,r) • hold(cr,k) • hold(rt,q)
                * G(r,k,q) + sum
        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 = -0.5 * [(x-m_k)^T -1 (x-m_k)] * .5
    return e^expo

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

1981 Machine Processing of Remotely Sensed Data Symposium
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.