Reprinted from

Tenth International Symposium

Machine Processing of

Remotely Sensed Data

with special emphasis on

Thematic Mapper Data and

Geographic Information Systems

June 12 - 14, 1984

Proceedings

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

MULTIRESOLUTION SPATIALLY CONSTRAINED CLUSTERING OF REMOTELY SENSED DATA ON THE MASSIVELY PARALLEL PROCESSOR

J.C. TILTON

Science Applications Research
Riverdale, Maryland

ABSTRACT

The improved spatial resolution of the Landsat Thematic Mapper (TM) and other new earth resources satellites increases the need for effective utilization of spatial information in machine processing of remotely sensed data. In the past it has proved difficult to implement algorithms exploiting spatial information because of the computational and organizational constraints imposed by the available serial computers. Fortunately, recently developed parallel computers, such as the Massively Parallel Processor (MPP) at NASA's Goddard Space Flight Center, are well suited for algorithms that exploit spatial information through utilizing neighborhoods of pixels. In an earlier paper, we explored region growing on the MPP as one promising technique for utilizing spatial information. In this paper, we discuss a multiresolution spatially-constrained clustering (MSCC) algorithm which we are implementing on the MPP. The MSCC algorithm uses a coarse-to-fine resolution schema which has as its core a region growing algorithm similar to the one described earlier and region switching and region splitting algorithms which are described in this paper. Following a detailed discussion of the MSCC algorithm we consider some applications of the algorithm.

I. INTRODUCTION

The improved spatial resolution of the Landsat Thematic Mapper (TM) and other new earth resources satellites increases the need for effective utilization of spatial information in machine processing of remotely sensed data. Using conventional per pixel algorithms, several studies have shown that the 30-meter resolution TM data tends to produce poorer analysis results than the 80-meter resolution MSS data. (Markham and Townshend, 1981; Williams et al, 1983; Alexander et al, 1983) It is generally agreed that a main reason why TM data produces such poor results with per pixel algorithms is these algorithms do not use the spatial information contained in the TM data to resolve confusions caused by the greater spatial detail.

In the past it has proved difficult to implement algorithms exploiting spatial information because of the computational and organizational constraints imposed by the available serial computers. Fortunately, recently developed parallel computers are well suited for algorithms that exploit spatial information through utilizing neighborhoods of pixels. One such computer is the Massively Parallel Processor (MPP), which was delivered in May 1983 by Goodyear Aerospace Corporation to NASA's Goddard Space Flight Center. (Batcher, 1980; Schaefer et al, 1982) The MPP is a Single Instruction, Multiple Data stream (SIMD) computer containing 16,384 bit serial microprocessors logically connected in a 128-by-128 array with each element having direct data transfer interconnections with its four nearest neighbors.

In an earlier paper, we explored region growing on the MPP as one promising technique for utilizing spatial information in machine processing of remotely sensed data. (Tilton and Cox, 1983) In this paper, we discuss a multiresolution spatially-constrained clustering (MSCC) algorithm which we are implementing on the MPP. The MSCC algorithm uses a coarse-to-fine resolution schema which has as its core a region growing algorithm similar to the one described earlier along with region switching and region splitting algorithms

which are described in this paper. Later in this paper we will give a description of the overall design of the MSCC algorithm followed by a detailed discussion of each major step. Following this discussion we consider some applications of the algorithm.

Before we discuss our multiresolution SCC algorithm in more detail, we briefly describe Parallel Pascal, the language we used for implementing the MSCC algorithm on the MPP. This high level language for the MPP greatly facilitated the development of our algorithm.

II. PARALLEL PASCAL

In order to fully exploit the capabilities of the MPP and make it a useful tool to not only the image processing researcher, but also the remote sensing applications scientist, the MPP developers placed a high priority on the development of a high level language to be used on the MPP. The first high level language developed, known as Parallel Pascal, is an extended version of the popular Pascal programming language. Developed by Reeves and Bruner* at Purdue University, Parallel Pascal was designed to be easy to use, portable, efficient, and was designed to provide good error detection. (Reeves and Bruner, 1980) The parallel extensions are a small number of carefully chosen features with an eye toward making Parallel Pascal portable even among the wide variety of parallel architectures. These extensions to Pascal are few and straightforward to allow the declaraton of parallel array, a few parallel manipulation and indexing functions and extended control structures.

Parallel arrays are explicitly declared in Parallel Pascal by affixing the word PARALLEL to the standard Pascal array declaration, viz.:

name: PARALLEL ARRAY[index range] OF type

This explicit declaration of parallel arrays provides the programmer direct control over which arrays are operated on in parallel. The last two dimensions of parallel arrays are restricted in Parallel Pascal to 128-by-128 (or the last dimension to 16,384) to conform with the architecture of the MPP.

Operationally, parallel arrays are different from standard Pascal arrays in that parallel arrays can be added, divided, compared, etc. as aggregate units rather than element-by-element. Special indexing mechanisms are provided to selectively operate on particular portions of a parallel array. Standard elemental functions (see Table 1 of (Tilton and Cox, 1984)) perform the same operation on each element of a parallel array, independently and in parallel. Special transformational functions (see Table 2 of (Tilton and Cox, 1984)) perform transformations upon the entire parallel array. We will introduce some of these functions below in the discussion of the MSCC algorithm.

III. GENERAL DESCRIPTION OF OUR APPROACH

The general design for the MSCC algorithm is as follows:

1. Initialize the clustering by segmenting the image into n-by-n subregions, where n>>2.
2. Cluster the data using iterative parallel region growing and region switching.
3. If the current clustering is done with the finest resolution subregions (usually n=2), go to step 7. Otherwise continue to the next step.
4. Generate a finer resolution set of subregions from the data (e.g. let $n'=n/2$) and initialize the region labeling of these subregions with the region labels from the previous clustering.
5. Use a region-splitting process to split out regions that were not resolved in the coarser resolution clustering.
6. Return to step 2.
7. Refine region edges by allowing individual pixels in region edge subregions to switch to a neighboring region where appropriate.
8. Stop.

Sections IV through VIII below discuss in more detail the major steps of the MSCC algorithm.

---

* A. P. Reeves is now with the School of Electrical Engineering, Cornell University, and J. D. Bruner is now with the Lawrence Livermore Laboratory.

IV. INITIALIZATION

The MSCC algorithm is initialized by segmenting the image into n-by-n subregions. For a p-by-p pixel image, n should be chosen to be the larger of 2 or $\lceil p/128 \rceil$, where $\lceil x \rceil$ designates the "ceiling" of x, i.e., the next larger integer (we consider square images for convenience; the generalization to rectangular images is not difficult). Such a choice of n most efficiently utilizes the MPP microprocessor array and brings the whole image into the array, albeit at a coarse resolution if n >> 2. An m-by-m array of subregions (each sized n-by-n pixels) is formed from the p-by-p data set, where $\lfloor m=p/n \rfloor$ and $\lfloor x \rfloor$ is the "floor" of x (the next smaller integer). (We throw away a small portion of data - less than n lines and/or columns - if n does not divide p evenly.) We then load the initial subregion feature values directly into the MPP's 128-by-128 array of microprocessors, one subregion per microprocessor. The region feature values we use are mean, standard deviation and degrees of freedom. In addition, a region label is given to each initial subregion. The label values given to each initial subregion are calculated based on the row (i) and column (j) of the microprocessor the region feature values are loaded into. We use the label values (i-1) + (j-1)*m; i=1,2,...,m; j=1,2,..., m. (If m<128, we load the initial region feature value into the upper left m-by-m array of MPP microprocessors and place impossible values into the remaining locations, e. g. negative values are impossible for all feature values and values >=m**2 are impossible region degree of freedom feature values.)

We now have initial subregion mean, standard deviation, degree of freedom and label values loaded into the 128-by-128 MPP microprocessor array. These variables can be declared in Parallel Pascal as:

```
mean:  PARALLEL ARRAY[1..128, 1..128]
       OF REAL;
stdev: PARALLEL ARRAY[1..128, 1..128]
       OF REAL;
dof:   PARALLEL ARRAY[1..128, 1..128]
       OF INTEGER;
label: PARALLEL ARRAY[1..128, 1..128]
       OF INTEGER;
```

Now that the algorithm initialization is complete, we go to the second step: iterative parallel region growing and region switching.

V. REGION GROWING AND REGION SWITCHING

Alternate iterations of parallel region growing and parallel region switching form the core of our MSCC algorithm. First we invoke region growing. Here all neighboring regions are compared (in parallel) in terms of a predefined similarity criterion, and the most similar pair of regions in the entire region are merged. Similarity criteria for all neighboring regions are recomputed and the most similar pair of regions are again merged. After a preset number of region growing iterations, region switching is invoked. Region growing and region switching are then alternated until a preset minimum number of regions is reached, or a preset minimum similarity criterion is met.

We now discuss parallel region growing and parallel region switching in turn. We also give portions of the Parallel Pascal program for parallel region growing, as an example of programming in Parallel Pascal.

A. ITERATIVE PARALLEL REGION GROWING

The first step of parallel region growing is to compare (in parallel) for each region each neighboring region in terms of a predefined similarity criterion. This is done in Parallel Pascal by declaring as parallel arrays "neighbor" feature value nbmean, nbstdev, nbdof and nblabel variables (as above) and shifting the mean, stdev, dof and label values from the specified neighboring microprocessor into the corresponding neighbor feature value parallel array. For example, the following Parallel Pascal statements "rotate" the array values one microprocessor from right to left ("east" neighbor) for all microprocessors in parallel:

```
nbdof := rotate(dof, 0, -1) ;
nbmean := rotate(mean, 0, -1) ;
nbstdev := rotate(stdev, 0, -1) ;
nblabel := rotate(label,0,-1) ;
```

The rotate function circularly rotates the data within the parallel array in a specified direction. We use the rotate function rather than the shift function (which does an end-off shift of the data) because of efficiency considerations. We look at the "east" neighbor first, followed by the "southeast," "south," and "southwest" neighbors. (We don't have to look "west," "northwest," "north" and "northeast" because a look "west" is redundant with a look "east," etc.) Using rotate, we can bring the

"southeast" neighbor into each microprocessor memory by using one rotate of the "east" neighbor values to the "north", rather than by two shifts of the original feature values: one to the "west" followed by one to the "north." The rotate of "east" neighbor values to the "north" is accomplished in Parallel Pascal as follows:

```
nbdof := rotate(nbdof, -1, 0) ;
nbmean := rotate(nbmean, -1, 0) ;
nbstdev := rotate(nbstdev, -1, 0) ;
nblabel := rotate(nblabel, -1, 0) ;
```

The "south" neighbor is obtained through an "east" rotation of the "southeast" neighbor values, and the "southwest" neighbor is obtained through an "east" rotation of the "south" neighbor values.

For each neighbor, the similarity criterion is calculated for all microprocessors in parallel through a function call:

```
compare(mdir,dof,nbdof,mean,nbmean,
              stdev,nbstdev,testval) ;
```

where mdir is a scalar variable designating which neighbor is being considered, and testval is the result of the similarity criterion calculation. The similarity criterion we use is the geometric mean (or, equivalently, the product) of the probabilities of random occurrence of the t- and F-statistics calculated from a modified Student's t-test and modified F-ratio test, respectively. (Tilton and Cox, 1983, p.132)

Some of the microprocessors at the edge of the image will contained invalid values for testval. We flag these microprocessors by storing impossible degree of freedom values in those locations (dof = m*m). The value of testval is set to zero for these microprocessors using the following Parallel Pascal "where" statement:

```
where dof > maxdof do testval := 0.0 ;
```

(The compare function itself zeros the testval values appropriately at the edge of the MPP microprocessor array.) Neighboring pixels that are already in the same region are eliminated by the following statement:

```
where nblabel = label do testval := 0.0 ;
```

We record the best comparison value from testval and the corresponding merge direction from mdir in the parallel arrays bestval and bestmdir, respectively. We initialize bestval and bestmdir when mdir=1 (corresponding to the "east" neighbor) with the following parallel assignment statements:

```
bestval := testval ;
bestmdir := mdir ;
```

For mdir = 2, 3 and 4 (corresponding to the "southeast," "south," and "southwest," neighbors) the values of bestval and bestmdir are updated with the following pair of statements:

```
where testval > bestval do
              bestmdir := mdir ;
where bestmdir = mdir do
              bestval := testval ;
```

The best merge for the entire image is then found by merging the pair of regions associated with the overall best comparison value in the bestval array. These best pair regions are found with the following statements:

```
maxtval := max(bestval,1,2) ;
if maxtval >= threshval do
  begin  (* find best pair of regions *)
    mask := 0 ;
    where bestval = maxtval do mask:=1 ;
    mdir := max(mask*bestmdir,1,2) ;
    case mdir of
    1: begin
         nblabel := shift(label,0,-1) ;
       end ;
    2: begin
         nblabel := shift(label,-1,-1) ;
       end ;
    3: begin
         nblabel := shift(label,-1,0) ;
       end ;
    4: begin
         nblabel := shift(label,-1,1) ;
       end ;
    end ;
    rlabel := max(mask*label,1,2) ;
    nbrlabel := max(mask*nblabel,1,2) ;
  end
else
  begin  (* exit subroutine *)
                    .
                    .
                    .
  end
end ;
```

The max(array,1,2) subroutine returns the maximum value from the specified two-dimensional parallel array. The subroutine is exited if the maximum best comparison test value is less than a preset threshold value (threshval).

The specified pair of regions (rlabel and nbrlabel) are merged with the following Parallel Pascal statements:

```
(* Extract the feature values for the
   regions to be merged. *)

   mask := 0 ;
   where label = rlabel do mask := 1 ;
   vdof[1] := max(mask*dof,1,2) ;
   vmean[1] := max(mask*mean,1,2) ;
   vstdev[1] := max(mask*stdev,1,2) ;
   mask := 0 ;
   where label = nbrlabel do mask := 1 ;
   vdof[2] := max(mask*dof,1,2) ;
   vmean[2] := max(mask*mean,1,2) ;
   vstdev[2] := max(mask*stdev,1,2) ;

   (* calculate feature values for new
      region *)

   vdof[0] := vdof[1] + vdof[2] + 1 ;
   vmean[0] := (vdof[1]+1)*vmean[1]
            + (vdof[2]+1)*vmean[2] ;
   vmean[0] := vmean[0]/(vdof[0]+1) ;
   vstdev[0] := vdof[1]*sqr(vstdev[1])
            + (vdof[1]+1)*sqr(vmean[1])
            + vdof[2]*sqr(vstdev[2])
            + (vdof[2]+1)*sqr(vmean[2])
            - (vdof[0]+1)*sqr(vmean[0]);
   vstdev[0] := sqrt(vstdev[0]/vdof[0]);

   (* Update new region *)

   where label = nbrlabel do
                 label := rlabel ;

   where label = rlabel do
     begin
       dof := vdof[0] ;
       mean := vmean[0] ;
       stdev := vstdev[0] ;
     end ;
```

Note that vdof, vmean and vstdev are 3 element "serial" vectors. The region merging process itself is largely a serial operation, whereas finding which pair of regions to merge is a highly parallel operation.

B. ITERATIVE PARALLEL REGION SWITCHING

In parallel region switching all original subregions are compared to the region to which they currently belong and to any neighboring regions (subregions internal to a region have no neighboring region, and thus cannot participate in region switching). (After initialization above, we stored the original subregion feature values in the parallel arrays omean, ostdev, odof and olabel.) The comparison with the current region is done with the subregion temporarily removed from the region. The region switching criterion is the difference between the subregion comparison with the neighboring region and the subregion comparison with the current region. The subregion with the largest region switching criterion value is then switched from its current region to the indicated neighboring region. The region switching criterion is recalculated for all subregions, and the region switching process is repeated iteratively until no more region switches occur. Then one more region iteration is performed, after which region switching is again invoked. Region switching is always continued until no more region switches occur.

Region switching is needed because as a region grows, its feature values may gradually become very different from its original feature values. When this occurs, the region feature values may become sufficiently different from the feature values of certain subregions making up the region that these subregions may become more similar to some adjacent region. The region switching process switches these subregions over to the appropriate neighboring region.

VI. TRANSITION TO CLUSTERING AT FINER RESOLUTION

After the region growing and region switching process converges, we generate a finer resolution set of subregions from the original data and initialize the clustering of these subregions with the region cluster map formed from the previous coarser resolution set of subregions. This is the multiresolution portion of the MSCC algorithm. If the previous set of subregions were of size n-by-n, the new finer set of subregions will be of size n'-by-n' with $n' < n$. For example, we could have $n'=n/2$.

We do multiresolution spatially constrained clustering rather than single-resolution spatially constrained clustering for two related reasons. Starting the MSCC algorithm with subregions sized so that the entire data set can fit into the 128-by-128 MPP microprocessor array greatly facilitates piecing together clusterings of the entire data set if the data set is larger than 256-by-256 pixels. The piecing together process is facilitated by this because of the second reason we use multiple resolutions: the coarser resolution clusterings initialize the finer resolution clusterings with global information about the data set. This global information serves to provide direction to the finer resolution clusterings and should substantially improve the overall quality of the final cluster map. It also tends to make each section of the data set to have

clusterings that are more consistent with each other at the boundaries.

VII. REGION SPLITTING

When we go from a coarser resolution set of subregions to a finer resclution set, certain smaller scale features may appear in the finer resolution set of subregions that were not detectable in the coarser resolution set. Because of this, we need to be able to split new regions out from the set of cluster regions obtained at coarser resolution.

Our region-splitting approach is quite similar to our region switching approach. For region splitting, each n´-by-n´ subregion is compared to the cluster region it is currently assigned to. The subregion that is most unlike its current cluster region is found, and if the comparison test value is less than a preset threshold, the subregion is split out of its current cluster region to form a new region. The subregion to current cluster region comparisons are repeated, and the most dissimilar subregions are split out until no subregion is dissimilar enough to pass the threshold test.

VIII. FINAL REGION EDGE REFINEMENT

After region splitting the MSCC algorithm returns to iterative parallel region growing and region switching (n now equals n´). If n is greater than some minimum (usually n=2), a finer resolution set of subregions is again generated and region-splitting is again invoked. If n is equal to the preset minimum value, after a round of region growing and switching, the MSCC algorithm continues on to the step described in this section: final region edge refinement.

Region edge refinement is actually region switching done on 1-by-1 "subregions." Since 1 pixel "subregions" cannot have a standard deviation, the region edge refinement comparison test reduces to a comparison of grey-scale value of each "subregion" to the mean of its current cluster region and the mean of each neighboring cluster region. (Pixels not on cluster region edges cannot participate in region edge refinement.) We use the following comparison test. Let the parallel array omean hold the the grey-scale values of each 1-by-1 "subregion," and let mean contain the mean value of the current cluster regions. (The omean parallel array is larger than 128-by-128, but by holding four values in each MPP microprocessor, an omean parallel array as large as 256-by-256 can be stored in the MPP array.) Let nbmean contain the mean values of a neighboring region in a particular direction ("east", "southeast," etc.). The comparison test is:

```
testval = ABS(mean-omean)
     + SIGN(mean-omean)*ABS(nbmean-omean)
```

After this comparison test is calculated for all neighboring regions, the 1-by-1 "subregion" with the largest comparison test value is switched to the appropriate neighboring region, if that largest test value is larger than a preset threshold value (e.g. threshold = 0.0). The comparison test are recalculated, and the "subregion" with the next largest comparison test is switched, etc., until no 1-by-1 "subregion" passes the threshold value test.

We now have the final region cluster map from the MSCC algorithm for the current portion of the data set. If the data set is larger than 256-by-256, the processing of the remaining portions of the data set must be completed and the region cluster maps must be stitched together. This stitching process will be the subject of a later paper.

IX. APPLICATIONS OF MULTIRESOLUTION SPATIALLY CONSTRAINED CLUSTERING

The MSCC algorithm can be used simply to produce segmentations of remotely sensed imagery. If a segmentation is the desired final result, one would most likely want to used very low threshold values in the various comparison tests in the algorithm. If the comparison test thresholds are too high, the resulting cluster map would contain too many regions to provide a useful segmentation.

The MSCC algorithm can also be used as a very effective "front-end" for a couple other algorithms. One is the ECHO classification algorithm. (Kettig and Landgrebe) ECHO runs in two stages: a segmentation stage and a sample classification stage. Revising ECHO to use the MSCC algorithm in the segmentation stage should substantially improve the performance of the ECHO algorithm.

The Cluster Compression Algorithm (CCA) is the other algorithm for which the MSCC algorithm can serve as a very

effective "front-end." (Hilbert, 1977) The CCA operates by clustering data over blocks (e.g. 16-by-16 pixels or 8-by-8 pixels) and storing the cluster map for each block and the mean values of the clusters in each block. Data compression factors of 10 or more can be obtained using this approach. The CCA could be modified to use the MSCC algorithm to provide a cluster map and cluster means for the entire data set. A grid (e.g. 16-by-16 or 8-by-8 pixels) could then be imposed on the cluster map, and the cluster region labels could be renumbered in each resulting cluster map block. Then the cluster map for each block and the mean values of the clusters in each block can be stored as before. In this case, however, there would be a variable number of clusters in each block and the amount of compression would depend on the thresholds set in the MSCC algorithm and on the data set itself.

X. FINAL REMARKS

The MSCC algorithm is currently designed to process single-band images. However, the algorithm is restricted to single-band images only by the comparison tests currently used in the region growing, switching and splitting processes and in the final cluster region refinement process. Generalizing these comparison tests to multi-band tests would allow the MSCC algorithm to process multi-band images. The easiest way to do this is to change the tests from single-band (univariate) tests to multiple-univariate tests. It would be more difficult to make the comparison tests into multivariate tests. We will be looking into this problem in the future.

The MSCC algorithm uses the full power of the MPP in the portions of the algorithm that calculate comparison tests between neighboring regions. The actual merging of a pair of regions, the switching of region assignment, or the splitting out of a region are substantially serial operations, however. Nevertheless, the MSCC algorithm is projected to run significantly faster on the MPP than on any serial processor. (Firm estimates of comparable execution times were not available at this writing.)

Multiresolution Spatially-Constrained Clustering (MSCC) offers one way of utilizing spatial information in the analysis of remotely sensed imagery data. Without the Massively Parallel Processor (MPP) it would be impractical to run the MSCC process on anything but a very small data set. Implemented on the MPP, or another similar large parallel computer, the MSCC algorithm becomes an effective method for segmenting remotely sensed data using a spatially-constrained clustering process.

XI. REFERENCES

Alexander, D., et. al., "Spectral, Spatial and Radiometric Factors in Cover Type Discrimination," Proceedings of the 1983 International Geoscience and Remote Sensing Symposium, San Francisco, CA (1983).

Batcher, K.E., "Design of a Massively Parallel Processor," IEEE Transactions on Computers, Vol. C-29, pp. 836-840 (1980).

Burkley, J.T., C.T. Mickelson, "MPP: A Case Study of a Highly Parallel System," Proceedings of the AIAA Conference on Computers in Aerospace #4, (1983).

Hilbert, E.E., "Cluster Compression Algorithm: A Joint Clustering/Data Compression Concept," JPL Publication 77-43, NASA Jet Propulsion Laboratory, Pasadena, California (1977).

Kettig, R.L. and D.A. Landgrebe, "Classification of Multispectral Image Data by Extraction and Classification of Homogeneous Objects," IEEE Transactions on Geoscience Electronics, Vol. GE-10, No. 1, pp. 19-26 (1976).

Markham, B.L. and J.R.G. Townshend, "Land Cover Classification Accuracy as a Function of Sensor Spatial Resolution," Proceedings of the Fifteenth International Symposium on Remote Sensing of Environment, Ann Arbor, MI (1981).

Reeves, A.P., and J.D. Bruner, "High Level Language Specification and Efficient Function Implementation for the Massively Parallel Processor," Interim Report TR-EE 80-32, School of Electrical Engineering, Purdue University, West Lafayette, IN (1980).

Reeves, A.P., and J.D. Bruner, "The Language Parallel Pascal and Other Aspects of the Massively Parallel Processor," Final Report for NASA Grant 5-3, School of Electrical Engineering, Cornell University, Ithaca, NY (1982).

Schaefer, D.H., J.R. Fischer, K.R. Wallgren, "The Massively Parallel Processor," AIAA Journal of Guidance, Control, and Dynamics, Vol. 5, No. 3,

pp. 313-315 (1982).

Tilton, J.C., and S.C. Cox, "Segmentation of Remotely Sensed Data Using Parallel Region Growing," Proceedings of the Ninth International Symposium on Machine Processing of Remotely Sensed Data, West Lafayette, IN (1983).

Tilton, J.C. and J.P. Strong, "Analyzing Remotely Sensed Data on the Massively Parallel Processor," Proceedings of the Tenth International Symposium on Machine Processing of Remotely Sensed Data, West Lafayette, IN (1984).

Williams, D.L., et. al., "Impact of Thematic Mapper Sensor Characteristics on Classification Accuracy," Proceedings of the 1983 International Geoscience and Remote Sensing Symposium, San Francisco, CA (1983).

James C. Tilton is a Senior Scientist with Science Applications Research (SAR), Riverdale, MD. He received a B.A. in Electrical Engineering, Environmental Science and Engineering and Anthropology from Rice University in 1976; M.S.E.E. from Rice University also in 1976; a M.S. in Optical Sciences from the University of Arizona in 1978 and a Ph.D. in Electrical Engineering from Purdue University in 1981. Since 1982 Dr. Tilton has been conducting and directing research in remote sensing data analysis techniques as a contractor at the NASA Goddard Space Flight Center. His research interests involve the development of optimal techniques for analyzing remotely sensed data. In particular this includes investigating techniques for incorporating spatial information into the analysis process and applying artificial intelligence techniques to the understanding of remotely sensed imagery.