CHAPTER 3

MORPHOLOGICAL ENDMEMBER IDENTIFICATION AND ITS SYSTOLIC ARRAY DESIGN

P.L. Aguilar, A. Plaza, P. Martínez, R.M. Pérez
Departamento de Informática, Universidad de Extremadura
Campus Universitario s/n 10071 Cáceres, Spain
E-mail: paguilar,aplaza,pablomar,rosapere@unex.es

Systolic arrays can be used in many different applications in order to improve performance. In particular, digital image processing algorithms are suitable to be implemented by systolic structures, since basic image manipulation operations are usually repetitive and can be mapped into a rectangular systolic structure. In this chapter we discuss the application of systolic arrays to speed up the performance of a new algorithm that performs unsupervised analysis of remote sensing images with high dimensionality (hyperspectral images). The methodology is based on mathematical morphology, which is a non-linear image analysis and pattern recognition technique that has proved to be especially well suited to segment images with irregular and complex shapes, but has rarely been applied to the classification/segmentation of hyperspectral images. The proposed method, which integrates spectral and spatial information in the analysis process, is fully described in this chapter, along with its hardware implementation by systolic arrays. A comparison between the hardware implementation and the software-based implementation is addressed.

3.1. Introduction

High-performance special-purpose computer systems are used to meet specific application requirements. As hardware cost decreases exponentially and the scale of integration in digital circuits is ever growing, the development of those circuits is obvious for special-purpose applications that demand large processing requirements [1].

Systolic arrays benefit from advances in semiconductor technology to provide adequate results in the case of applications requiring optimal throughput.

In terms of configuration, systolic arrays derive from other array-like structures, such as iterative arrays, cellular automata and processor arrays. These architectures are based on regular and modular formations that match the computational needs of many algorithms.

There exist several applications which include algorithms and processing modules appropriate for implementation by systolic arrays, such as digital image and signal processing, pattern recognition, linear and dynamic programming, etc. In fact, many of these require special-purpose implementations when used in real-time environments.

When systolic arrays were originally proposed, they were intended for applications requesting optimal throughput and large processing bandwidth, usually at the cost of response time. These applications can usually be supported by array-like structures consisting of a few types of single-processing units.

This chapter introduces an example of the applications mentioned above. In particular, we propose a systolic array structure that speeds up performance in a brand-new methodology; this framework serves to conduct an unsupervised analysis of multispectral or hyperspectral images. The use of systolic arrays in this specific application is very appropriate and illustrative of a practical case, as it demonstrates how basic image handling tasks can be mapped onto a rectangular systolic structure that takes advantage of the parallel processing capabilities provided by VLSI/ULSI technologies. We thus describe this methodology and its corresponding implementation by working with systolic arrays.

3.2. Overview of the Proposed Application

During the past decade, a number of airborne and satellite hyperspectral sensors have been developed or improved for remote sensing applications [2], [3], [4]. Image spectrometry enables the detection of materials, objects and regions in a particular scene with great accuracy. Hyperspectral data typically consist of hundreds of thousands of spectra, and, as a result, the analysis of this information is a key issue [5].

A very useful and commonly accepted approach to analyze hyperspectral data has been the identification of the purest spectra or

"endmembers" [6]. Some researchers have taken up the chore of constructing spectral libraries of pure elements that can be matched with every spectrum in a hyperspectral image in order to classify the scene [7]. This processing method is suitable when pure materials, contained in the library, are at ground level, but in real-world situations, only the strongest features are matched, mainly due to the fact that the materials are spatially or intimately mixed. In this sense, the most common technique is to determine endmember spectra directly from the image. Once the individual endmembers have been identified, several methods can be used to map their spatial distributions, associations and abundances [8], [9].

A wide variety of methodologies has been proposed in the literature in order to find endmembers in data cubes [10]. One of the most successful approaches is the Pixel Purity IndexTM (PPI) algorithm [11], which finds the "purest" pixels in the scene through a series of repeated projections onto randomly oriented lines in the N-dimensional space. These potential endmember spectra are loaded into an N-dimensional scatterplot and rotated in real time until a trained analyst selects extremities within the data cloud; these are likely to match scene endmembers. The procedure, based on the geometry of convex sets, is well accepted (made available through a commercial software system called ENVI®, standing for Environment for Visualizing Images), but it is time-consuming and highly interactive.

Several methods for autonomous extraction of endmembers have been recently proposed in the literature. The N-FINDR algorithm [10] finds the simplex with a maximum volume to be enclosed within all the points of the data cloud. The ORASIS algorithm [12] uses a process called Exemplar Selection to reduce the data set by rejecting any "redundant" spectra (this requires the calculation of the angle between spectral vectors). The Iterative Error Analysis (IEA) approach [13] performs a series of constrained sieving tasks and chooses as endmembers those pixels that minimize errors in the unmixed image. In addition to these physical methods, there are several statistics-driven approaches that rely on clustering algorithms [14]. The major drawback of all previous methods is that they only consider the spectral information contained in data cubes. Spatial information has not been fully exploited yet, specially in unsupervised classification. The integration of both spatial and spectral information is becoming more relevant as the sensors used in spaceborne platforms tend to increase the spatial resolution [15].

Mathematical morphology theory [16] is a non-linear technique widely used for image analysis and pattern recognition. Although it is namely befitting for segmentation of binary or grayscale images that have irregular and complex shapes, its application in the classification/segmentation of multispectral or hyperspectral images has been rare [17], [18]. In this chapter, we discuss a brand-new automated methodology to find endmembers in hyperspectral data cubes by applying mathematical morphology.

3.3. Methodology

In this section, we provide some basic concepts about mathematical morphology theory; we also describe a schema to extend morphological operators to the hyperspectral domain. Finally, we propose an automated methodology, based on morphology, to extract endmembers from data cubes.

3.3.1. *Classical Mathematical Morphology*

Mathematical morphology theory [16] has become a productive tool for image analysis and pattern recognition. In binary morphology, images are represented as sets, in which foreground pixels are members of a set X and background pixels belong to the complementary set XC. The two basic operations of mathematical morphology consist in the transformation of an image by another set K, known as the structuring element. The shape and size of the structuring element determine the spatial characteristics of the resulting image. The two basic morphology operations, dilation and erosion, are defined respectively as follows:

$$X \oplus K = \{s \mid \mathrm{K}_s \cap \mathrm{X} \neq \varnothing\} \tag{3-1}$$

$$X \otimes K = \{s \mid K_s \subseteq \mathrm{X}\} \tag{3-2}$$

Erosion and dilation are said to be dual regarding complementation. We define the opening of X with respect to K as the following set:

$$X_K = (X \otimes K) \oplus K \tag{3-3}$$

Similarly, we attribute the closing of X regarding K to the following set:

$$X^K = (X \oplus K) \otimes K \tag{3-4}$$

The opening of X by K is thus defined in terms of an erosion followed by a dilation. Similarly, the closing of X by K is defined in terms of a dilation followed by an erosion.

The principles of mathematical morphology have also been extended to the grayscale image case [19]. Grayscale images are described as a grey level function *f(x,y)* on the points of the Euclidean 2-space. Grayscale dilation, erosion, opening and closing are respectively described as follows:

$$(f \oplus k)(x,y) = \underset{(s,t)\in k}{Max} \{f(x-s, y-t) + k(s,t)\} \tag{3-5}$$

$$(f \otimes k)(x,y) = \underset{(s,t)\in k}{Min} \{f(x+s, y+t) - k(s,t)\} \tag{3-6}$$

$$f_k(x,y) = ((f \otimes k) \oplus k)(x,y) \tag{3-7}$$

$$f^k(x,y) = ((f \oplus k) \otimes k)(x,y) \tag{3-8}$$

The expressions for grayscale dilation and erosion are markedly similar to the convolution integral that is often encountered in digital image processing. Here, adding and differentiation replace multiplication and minimization, whereas maximization replaces addition.

3.3.2. *Extending Mathematical Morphology to the N-Dimensional Space*

Mathematical morphology precision requires an algebraic structure T (complex lattice) so that:

T is induced by a (partial) ordering relation.

For any family of elements in T, there exists a major member as small as possible, called supremum, and a minor member as big as possible called the infimum [18].

In hyperspectral images, these two properties are absent because there is no natural means for a complete arrangement of multivariate pixels. Two main strategies have been considered in the solution of this problem as a result:

The first one consists of processing each channel of the hyperspectral image separately. This marginal approach is not satisfactory since it fails to account for the existing correlation among individual channels.

The second tactic is based on a sheer vector approach to process all the hyperspectral channels at the same time. This strategy requires the definition of a vector-driven organizational task that determines the supremum and the infimum of any family of N-dimensional vectors.

The definition of a vector ordering relation can be done as follows. Let $x_1, x_2,\ldots,x_n$ be the hyperspectral image pixels within a filtering window that represents the structuring element of a morphological operation. We can define a measurement of the dissimilarity between two of those pixels, x_a and x_b by:

$$dist(x_a, \mathrm{x_b}) = \cos^{-1}\left(\frac{x_a \cdot x_b}{\|x_a\| \cdot \|x_b\|}\right) \tag{3-9}$$

This is the angular distance function, expressed in radians. The following scalar quantity can be calculated to measure the global distance between a particular pixel x_i and a set x_j, j=1..m of neighboring pixels.

$$d_i = \sum_{j=1}^{m} dist(x_i, \mathrm{x_j}) \tag{3-10}$$

The supremum and infimum of a set of x_i, i=1..n hyperspectral pixels are respectively defined as:

$$\sup(x_i) = h = \arg\max_i d_i \tag{3-11}$$

$$\inf(x_i) = l = \arg\min_i d_i \tag{3-12}$$

Thus, it is easy to determine basic morphology operations such as erosion and dilation by maximum and minimum operations, as indicated in the subsection above. While h is the most singular pixel (spectra-based) in a spatial neighborhood, l is nothing else than the median of the pixels in the neighborhood, according to the classical vector median definition [18].

3.3.3. *Autonomous Morphological Endmember Extraction (AMEE)*

We propose a morphology-based automated algorithm to extract endmembers from hyperspectral images. The input for this process is the full hyperspectral image cube, with no previous dimensionality reduction or pre-processing. The procedure must examine the full dataset to find those pure pixels that can be used to describe the various mixed pixels in the scene. This is done through the following steps.

Let $\boldsymbol{X}$ be the original image and $\boldsymbol{K}$ a structuring element. We propose the following morphological operator to identify endmembers:

$$X\Phi K = dist\left(X, X^{K}\right) = dist\left(X, (X \oplus K) \oplus K\right) \tag{3-13}$$

This operator works as follows: First, a hyperspectral closing operation is applied to the original image; then, a measurement of the dissimilarity between each pixel of the original image and the correspondent pixel in the closed image is calculated in terms of the angular distance function.

It is important to emphasize that Φ extracts spectral information on a pixel-by-pixel scale, much like any other existing methods, but the process is driven by spatial information. As a result, spatial and spectral responses are simultaneously considered in the analysis.

Let $\boldsymbol{K}$ be a square-shaped, 3x3 structuring element. Fig. 3-1 illustrates the working procedure of the operator when applied to a target pixel $\boldsymbol{E}$ in the original image, and using $\boldsymbol{K}$ as a structuring element. First, sheer spectra-based hyperspectral pixels ($\boldsymbol{Q}$) in a spatial neighborhood of the target pixel are selected after projecting each pixel against all the spatial neighbors. This procedure is repeated for every pixel in $\boldsymbol{X}$, leading to a new image $\boldsymbol{X}\oplus\boldsymbol{K}$. From this image, only the pixels that provide a good representation of their neighbors are selected by the minimum operator, which in our case is equivalent to a vector median filtering. The pixels in $(\boldsymbol{X}\oplus\boldsymbol{K})\oplus\boldsymbol{K}$ match the definition of an endmember, i.e. a pure spectra pixel that can be used to describe several mixed pixels in the scene. Finally, dissimilarity between $\boldsymbol{E}$, corresponding to the target pixel in the original image, and $\boldsymbol{V}$, the pixel selected by the operator, is calculated to check whether the target pixel is likely to be an endmember.

The described operation is repeated by working with structuring elements that have an increasing size, so that detailed information of the spatial context associated with each hyperspectral pixel is obtained.

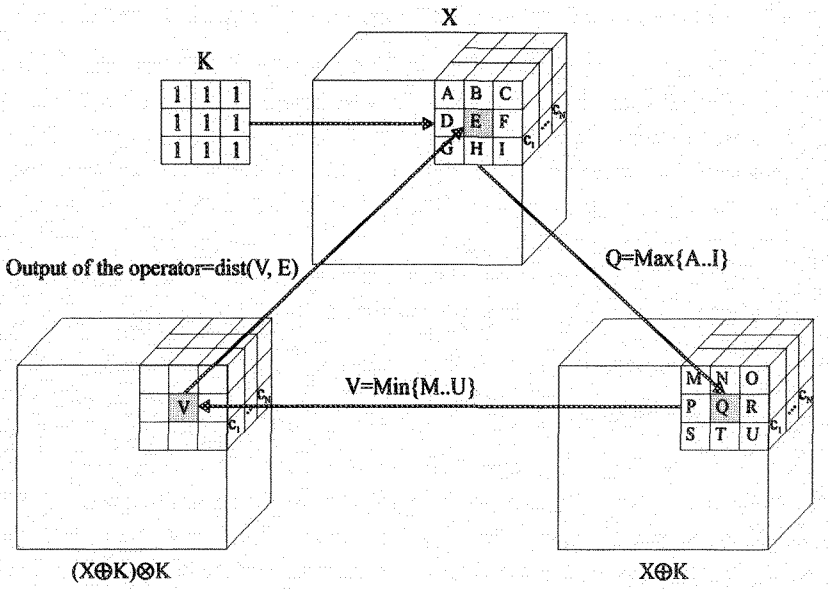

Fig. 3-1. Working procedure of the Φ hyperspectral morphological operator.

The overall result of applying a sequence of morphological operators to the original image is a spatial/spectral signature related to each pixel of the scene. From this information, a 2-dimensional grayscale rule image is generated by following the basic idea of the Pixel Purity Index algorithm: Those pixels that have been repeatedly selected by the operator during the process are more likely to be declared "pure". This image is automatically thresholded, and the resulting regions are developed by applying a spatial-spectral technique, as described in [20], to obtain the final endmembers. In our research, we focus on the hardware implementation of morphological eroding and dilating operations by means of systolic arrays, indicating the efficiency of these structures in the hardware optimization of a specific special-purpose software algorithm.

3.4. Systolic Array Implementation

The algorithm described above is characterized by the repetitive use of very simple operations with the input data. This fact implies that systolic arrays may provide an efficient VLSI implementation of this methodology. We also

give a detailed account of the operations of AMEE algorithms involved in systolic array design:

- Calculation of the spectral dissimilarity between two particular pixels in the hyperspectral image (i.e. x^i and x^j) by means of the angular distance function, expressed in Eq (3-5).
- Extraction of the purest spectra pixel from a set of observations. In order to perform this operation, as mentioned in Section 3.3, the definition of a vector ordering relation to determine the *supremum* and the *infimum* of any family of N-dimensional vectors is required. It is important to emphasize that each vector is associated with a particular hyperspectral pixel, and the family of vectors corresponds to a spatial neighborhood defined by the structuring element of the morphological operation (see Section 3.3). The calculation of the *supremum* and *infimum* can be done as follows:
  - ✓ Select one vector x^i related to a hyperspectral pixel.
  - ✓ Compute the angular distance between x^i and all the other vectors in the family, accumulating the partial results obtained (see Eq (3-6)). This calculation provides the cumulative distance between x^i and the rest of the vectors in the family.
  - ✓ Repeat Steps 1 and 2 with all the pixels in the spatial neighborhood. Therefore, we will have an accumulated distance value d_i per pixel.
  - ✓ According to Eq (3-7), determine the vectors having extreme values of d_i.

In the next subsections, we present a graph-dependent methodology to calculate the angular distance and the *supremum* and *infimum* by using systolic arrays.

3.4.1. *Rectangular Systolic Array to Calculate the Angular Distance*

Let $\boldsymbol{X}$ be a matrix containing all the pixels that belong to a certain spatial neighborhood, as defined by a structuring element.

$$X = \begin{bmatrix} \begin{bmatrix} x_1^1 & x_2^1 & \cdots & x_N^1 \end{bmatrix} \\ \begin{bmatrix} x_1^2 & x_2^2 & \cdots & x_N^2 \end{bmatrix} \\ \vdots \\ \begin{bmatrix} x_N^M & x_N^M & \cdots & x_N^M \end{bmatrix} \end{bmatrix}$$

The following pseudo-code algorithm is used to calculate the angular distance. M is the total number of pixels in the neighborhood. *Prod(i)* accumulates the outer product between pixel x^i and the rest of pixels in the neighborhood. *Norm_Xi* and *Norm_Xj* are the norms of pixels x^i and x^j, respectively. Finally, *dist(i)* is the accumulated distance of pixel x^i in relation to all the other pixels from the neighborhood.

Table 3-1. Set of indexes of the algorithm to calculate the outer product and vector norms.

i j k	Prod(i)	Norm_Xi(i)	Norm_Xj(i)	Xi(i,k)	Xj(j,k)
1 1 1	1	1	1	(1,1)	(1,1)
1 1 2	1	1	1	(1,2)	(1,2)
1 2 1	1	1	1	(1,1)	(2,1)
1 2 2	1	1	1	(1,2)	(2,2)
.....	.....	.....	.....	.....	.....
1 N 1	1	1	1	(1,1)	(1,1)
.....	.....	.....	.....	.....	.....
1 N N	1	1	1	(1,N)	(N,N)
.....	.....	.....	.....	.....	.....
2 1 2	2	2	2	(2,2)	(1,2)
2 2 1	2	2	2	(2,1)	(2,1)
2 2 2	2	2	2	(2,2)	(2,2)
.....	.....	.....	.....	.....	.....
M,1,1	M	M	M	(1,1)	(1,1)
.....	.....	.....	.....	.....	.....
M,N,N	M	M	M	(M,N)	(N,N

First, we proceed to the design of the systolic structure, corresponding to the first three iterations of this algorithm. From that initial design, we perform other operations needed in order to obtain the angular distance. Table 3-1 shows the set of indexes for this algorithm. In this table, each index element is shown as a three-tuple (i,j,m). Note that for both index elements $(i,j,1)$ and $(i,j,2)$, the same value of *Prod(i)* is used, i.e. *P(i)* can be piped onto k direction k. Similarly, values Xi(i,k) and Xj(j,k) can be piped onto directions j and i, respectively.

```
Begin
  For i = 1 to M
    For j = 1 to M
      For k = 1 to N
        Prod(i) = Prod (i) + x (i,k) * x (j,k)
        Norm_Xi (i) = Norm_Xa (i) + x (i,k) * x (i,k)
        Norm_Xj (i) = Norm_Xb (i) + x (j,k) * x (j,k)
      End For
    End For
  End For
  For i = 1 to M
    d (i) = Prod (i) / (Norm_Xi (i) * Norm_Xj (i) )
    dist (i) = cos-1 (d (i) )
  End For
End
```

Data-dependence vectors are found by equating indexes of all possible pairs of generated and used variables.

$d_1^T = (i, j, k) - (i - 1, j, k) = (1, 0, 0)$ to x(j,k)
$d_2^T = (i, j, k) - (i, j - 1, k) = (0, 1, 0)$ to x(i,k)
$d_3T = (i, j, k) - (i, j, k - 1) = (0, 0, 1)$ to prod(i) and norm_Xi, and norm_Xj

Fig. 3-2 shows the graph of dependences for the described algorithm variables. In this figure, we can observe the behaviour and movement of the variables: x^j moves from left to right, x^i moves from top to bottom, and the rest of the variables move onto direction j.

In order to map this algorithm with loops onto a systolic array, the approaches proposed by Kuhn [21] and Moldovan [22] are followed. The loop indexes of our algorithm are transformed into new loop indexes by a bijective and monotonic function T, which allows parallelism and pipelining, while preserving the dependences of the original algorithm.

$$T = \begin{bmatrix} \Pi \\ S \end{bmatrix} \qquad T = \begin{bmatrix} t_{11} & t_{12} & t_{13} \\ t_{21} & t_{22} & t_{23} \\ t_{31} & t_{32} & t_{33} \end{bmatrix}$$

where

$$\Pi = \begin{bmatrix} t_{11} & t_{12} & t_{13} \end{bmatrix} \qquad S = \begin{bmatrix} t_{21} & t_{22} & t_{23} \\ t_{31} & t_{32} & t_{33} \end{bmatrix}$$

In order to achieve a valid arrangement, and, at the same time, reduce the turnaround time (i.e. $\pi^* d_i > 0$), the mapping function π is selected as:

$$\Pi = [1 \quad 1 \quad 1]$$

Our choice of S determines the interconnection of the processors. We select S as:

$$S = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

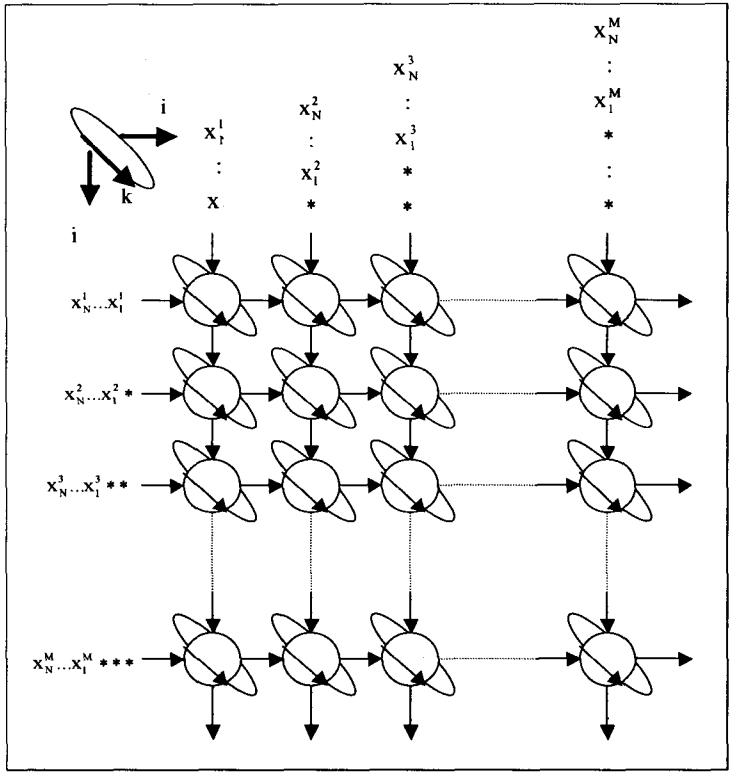

Fig. 3-2. Dependence graph for the obtaining the necessary computation the angular distance.

The advantage of this selection is that a projection onto direction k is performed. Thus, all the nodes aligned in direction k will be processed by the same processor element or PE. The interconnection function between processors is defined by:

$$Sd_i = \begin{bmatrix} x \\ y \end{bmatrix}$$

where x and y refer to the movement of the variable along directions j and i, respectively. In our case:

$$S{\cdot}d_1 = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} \bullet \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \qquad S{\cdot}d_{12} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} \bullet \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} S{\cdot}d_3 = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} \bullet \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

Table 3-2. Mapping the original index set onto the transformed index set.

I=(i j k)	T*I	Computations	*IT*I	*S*I
(1 1 1)	(3 1 1)	$Prod\_1\_1 = x_1^1 * x_1^1$ $Norm\_X1\_1 = x_1^1 * x_1^1$ $Norm\_X1\_1 = x_1^1 * x_1^1$	3	(1 1)
(1 1 2)	(4 1 1)	$Prod\_1\_1 = Prod\_1\_1 + x_2^1 * x_2^1$ $Norm\_X1\_1 = Norm\_X1\_1 + x_2^1 * x_2^1$ $Norm\_X1\_1 = Norm\_X1\_1\ x_2^1 * x_2^1$	4	(1 1)
.....	.....	.....	.....	.....
(1 1 N)	(N+2 1 1)	$Prod\_1\_1 = Prod\_1\_1 + x_N^1 * x_N^1$ $Norm\_X1\_1 = Norm\_X1\_1 + x_N^1 * x_N^1$ $Norm\_X1\_1 = Norm\_X1\_1\ x_N^1 * x_N^1$	N+2	(1,1)

Table 3-2 represents the mapping functions between the original and transformed (by *T*) indexes. In this table, columns *πxI* and *SxI* indicate the time and the cell processing the computation indexed by I.

From Table 3-2, we can conclude that a rectangular array formed by MxM processor elements (PEs) is needed. The number of clock cycles required to perform all the operations involved in the full calculation is 2M+N. Thus, in cycle N+3, the processing unit PE11 will have calculated the product between x_1 and its own, as well as its norm. One cycle afterwards, PE12 and PE21 will have performed the same operations for pixels x_1 and x_2. In this train of thought, we can conclude that by the cycle 1+N+M, the product between x_1 and all the pixels of the spatial neighborhood will have

been performed, and after 2M+N cycles, the operations associated with PEMM will have been finished.

Table 3-2 (cont.). Mapping the original index set onto the transformed index set.

I=(i j k)	T*I	Computations	Π^*I	S^*I
(1 2 1)	(4 2 1)	$Prod2\_1=x_1^1*x_1^2$ $Norm\_X1\_2=Norm\_X1\_2+x_1^1*x_1^1$ $Norm\_X2\_1=Norm\_X2\_1\ x_1^2*x_1^2$	4	(2 1)
(1 2 2)	(5 2 1)	$Prod2\_1=x_2^1*x_2^2$ $Norm\_X1\_2=Norm\_X1\_2+x_2^1*x_2^1$ $Norm\_X2\_1=Norm\_X2\_1\ x_2^2*x_2^2$	5	(2 1)
.....	.....	.....	.....	.....
(1 2 N)	(N+3 2 1)	$Prod2\_1=Prod2\_1+x_N^1*x_N^2$ $Norm\_X1\_2=Norm\_X1\_2+x_N^1*x_N^1$ $Norm\_X2\_1=Norm\_X2\_2\ x_N^2*x_N^2$	N+3	(2 1)
(2 1 1)	(4 1 2)	$Prod1\_2=x_1^2*x_1^1$ $Norm\_X2\_1=x_1^1*x_1^2$ $Norm\_X1\_2=x_1^1*x_1^1$	4	(1 2)
(2 1 2)	(5 1 2)	$Prod1\_2=Prod1\_2+x_2^2*x_2^1$ $Norm\_X2\_1=Norm\_X2\_1+x_2^2*x_2^2$ $Norm\_X1\_2=Norm\_X1\_2\ x_k^1*x_k^1$	5	(1 2)
.....	.....	.....	.....	.....
(2 j k)	(2+j+k j 2)	$Prodj\_2=Prodj\_2+x_k^j*x_k^2$ $Norm\_X2\_j=Norm\_X2\_j+x_k^j*x_k^j$ $Norm\_Xj\_2=Norm\_Xj\_2\ x_k^2*x_k^2$	2+j+k	(j 2)
.....	.....	.....	.....	.....
(i j,k)	(i+j+k k i)	$Prodj\_i=Prodj\_i+x_k^j*x_k^i$ $Norm\_Xi\_j=Norm\_Xi\_j+x_k^j*x_k^j$ $Norm\_Xj\_i=Norm\_Xj\_i\ x_k^i*x_k^i$	i+j+k	(k i)
.....	.....	.....	.....	.....
(M M N)	(2M+N M M)	$Prod_{M\_M}=Prod_{M\_M}+x_K^M x_K^M$ $Norm\_XM\_M=Norm\_XM\_M+x_K^M*x_K^M$ $Norm\_XM\_M=Norm\_XM\_M+x_K^M*x_K^M$	2M+N	(M M)

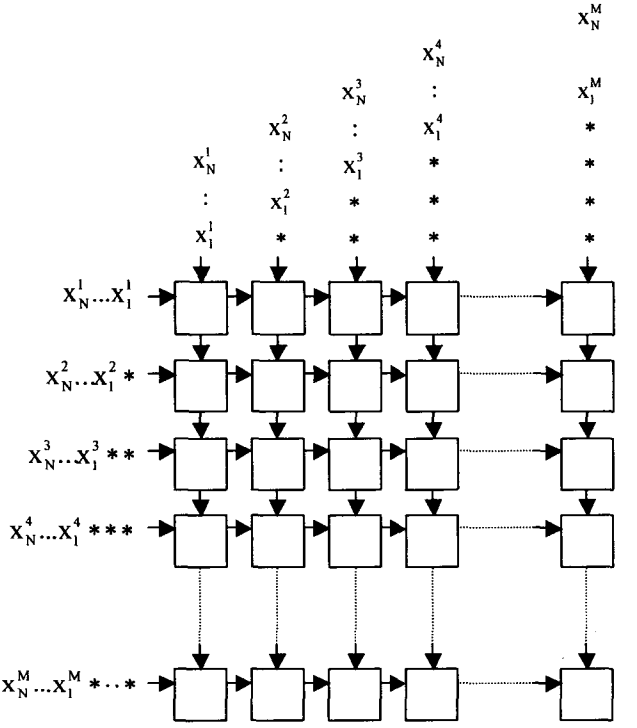

Fig. 3-3. Required interconnections between PEs to the systolic array for computing the outer-product and the norms of pixels.

Fig. 3-3 represents the required interconnections between the PEs. As shown in this figure, the resulting architecture has a rectangular array design. The pixels fed from left to right and top to bottom are transmitted between neighbouring PEs.

The capabilities of arithmetic processing related to each PE should include multiplication and accumulation operations (see Fig. 3-4). It is also required that each PE contains three accumulator cells to store the outer product and the norms of the involved pixels.

It is important to emphasize that AC1 refers to *Prod(i)*, AC2 to $norm\_x^j$ and AC3 to $norm\_x^j$. As described above, in cycle N+2, the processor element PE11 performs the operations associated with the calculation of variable *d(i)* and the angular distance; this is one of the objectives of our

algorithm. In subsequent cycles, the same operation will be performed for other PEs.

$$AC1 = \cos^{-1}\left(\frac{AC1}{AC2 * AC3}\right)$$

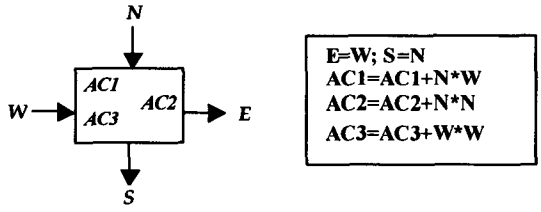

Fig. 3-4. Operations performed by Fig. 3-3 PEs.

This distance is stored in the PE because it will be needed in the determination of cumulative distances. If adding all the AC1 by rows and columns, we obtain as output the cumulative distance of each vector in relation to all the other vectors. Subsequently, the outputs of PE_{1M} and PE_{M1} have the same value, corresponding to the cumulative distance between pixel x1 and the rest of the pixels. The cumulative distance between x2 and the rest of the pixels can be obtained by the outputs of PE_{2M} and $PE_{M2,}$ and so on. In order to achieve the calculation of cumulative distances, it is necessary that, during the M cycles that follow the determination of the angular distance, each PE performs the operations:

$$S = N + AC1 \qquad E = W + AC1$$

It is important to note that, in this case, the input to the extreme left and at the very top of the initial elements must be 0. Thus, in order to obtain all the cumulative distances, M clock cycles must be added to the 2M+N cycles expressed in Table 3.2. From the previous exposition of ideas, we thus derive that, during those M cycles, the PEs perform operations that differ from those described in Fig. 3-4. Therefore, programmable PEs are required to obtain different functionalities at different clock cycles.

3.4.2. *Linear Systolic Array to obtain Supremum and Infimum of Cumulative Distances*

By following the procedure described in section 3, after obtaining the cumulative distances at each pixel of the neighborhood, we must obtain the pixels with the maximum and minimum associated cumulative distance values. The following simple algorithm can be used to realize such a computation.

```
Begin
        Infimum=∞
        Supremun=0
        For i = 1 to M
                If dist_acum(i) < infimum
                        Then Infimum = dist_acum(i)
                Endif
        Endfor
        For j=1 to M
                If dist_acum(j) >supremum
                        Then Supremun = dist_acum(j)
                Endif
                End For
End
```

In order to implement the algorithm, we propose the use of two linear arrays, one to calculate the supremum and another to calculate the infimum. Each one will have M processor elements respectively distributed in unique horizontal and vertical directions.

Fig. 3-5 represents the required interconnections between the PEs. As shown in this figure, the architecture has a linear array design. The supremum (or infimum) initial value is fed from the leftmost (or upmost) side of the array, and is transmitted between neighboring PEs.

The only operation performed at each PE is a comparison, and the resulting value is transmitted to the closest PE on the right or at the bottom. After M cycles, the last PE provides the maximum (or minimum) of the angular distances. The operations supported at each PE are shown in Fig. 3-6, where a) corresponds to the horizontal linear array and b) to the vertical array.

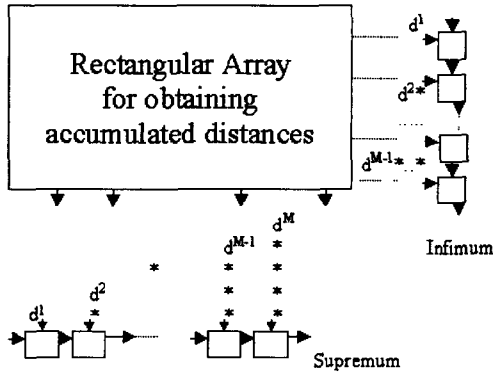

Fig. 3-5. Required interconnections between PEs for calculating the supremum and infimum values.

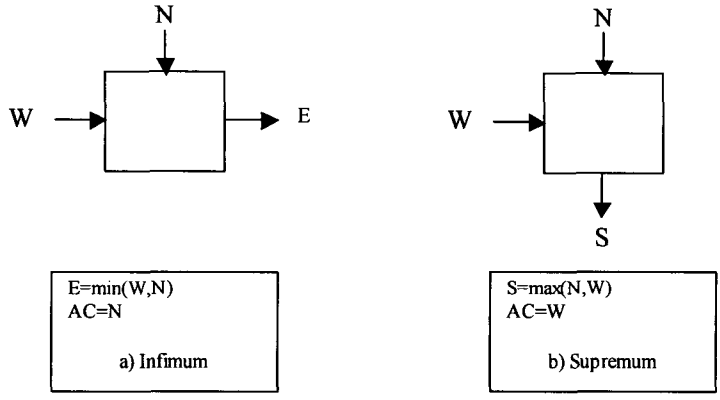

Fig. 3-6. Operations performed by Fig. 3-5 PEs of linear array.

The pixel associated with the maximum and minimum value of the cumulative distance is determined by a simple procedure: By providing feedback to the outputs of the linear arrays and comparing the outputs corresponding to each pixel, stored at each PE. The elements PEi of the horizontal array and PEj of the vertical array which are able to detect the matching, will indicate the desired extreme vectors x^i x^j. Those vectors have been stored in elements PE_{iM} and PE_{Mj} of the rectangular array when *Prod(i)* and *Prod(j)* were calculated. Fig. 3-7 shows a diagram of the systolic array

final design, including feedback connections between the PEs as well as the operations performed with each processing element.

3.5. Summary of Design

We deem that, at this stage, summing up the described systolic array is suitable. The AMEE model can be mapped onto a systolic structure formed by a rectangular array of MxM PEs and two linear arrays, each one consisting of M PEs. The PEs should be programmable due to the variable functionalities involved. In order to describe the design, we distinguish between data movements and PE design requirements.

Data movements into the rectangular systolic design

In this implementation, we have distinguished two main steps: First, the angular distance, and, second, cumulative distance computations. Next, we offer a detailed description of the steps required to perform each operation.

Angular Distance Computation

Data movements comprise:

- ✓ Pixel vector set ***X***, as input into the first row of the PE array, -- one column for each PE. They are subsequently propagated downwards to all PEs on the same column. This same set serves as input for the first column of the PE array. They are subsequently propagated rightward to all PEs on the same row.
- ✓ When the values of pixel vectors x_k^j and x_k^i arrive at the *ij*th PE (from the top and right respectively) both are multiplied by themselves and by it, giving way to partial sums *Prodi_j(k)*, *Norm_xi(k)* and *Norm_xj(k)*. These values are accumulated in the same EP in order to compute the angular distance, which will be used in determining the cumulative distance.

Cumulative distance determination

In this case, data movements comprise the following steps:

- ✓ Once angular distances are calculated, a null (0) value is introduced in the PEs on the first row and first columns of the array, following a time-pipelined scheme (i.e., 0-value is introduced in the first PE at time t, in the second PE at time t+1, and so forth).

- ✓ Each PE_{ij} (or PE_{ji}) performs the addition between the value received from its neighboring PE at the left (PE_{i-1j} or PE_{ij-1}) and the stored distance, sending the resulting value to its neighboring PE at the right (PE_{ij+1}) or bottom (PE_{i+1j}).
- ✓ As soon as the data have arrived at the last PE of each row or column (PE_{iM}, or PE_{Mj}), the data can be routed to the linear array in order to determine the extreme distance values.

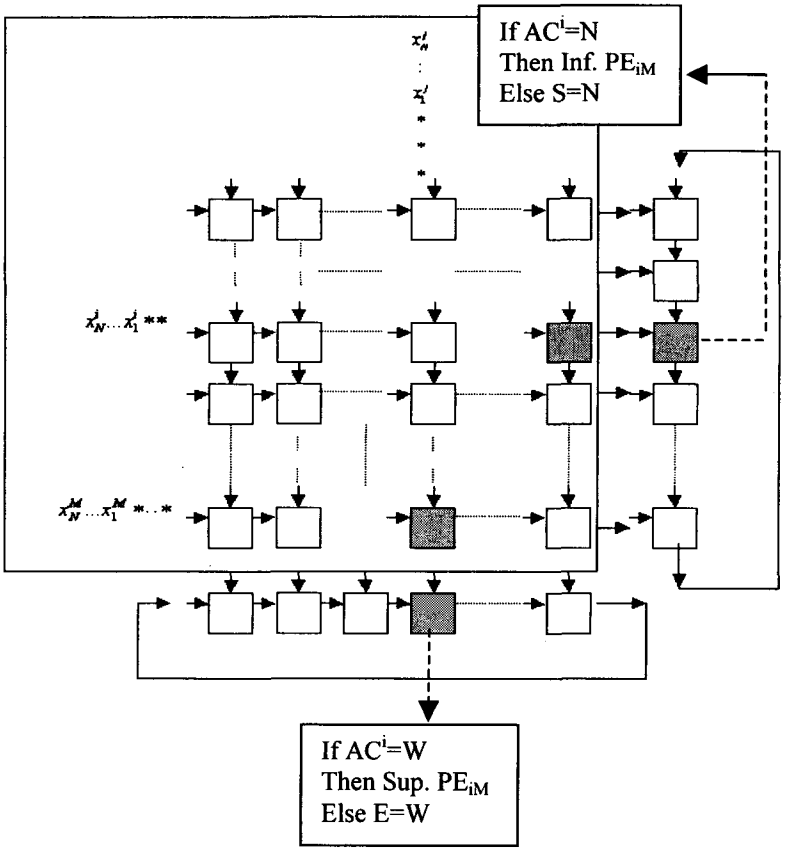

Fig. 3-7. Final systolic design scheme. PEs related to the calculation of extreme pixels are outlined.

Processor element design requirements into the rectangular systolic design

In this case, the processor elements comprise the following key components:

a) Each PE_{ij} should store a row of the set of pixel vectors $\boldsymbol{X}$, and the last element of each row and column must be able to store the pixel that arrives in a pipelined way from the left. Moreover, it is necessary that each PE incorporates three memory cells for storing the required computing.

b) Data are transmitted in two directions between neighbouring PEs, downwards and rightwards.

c) Each PE should support all arithmetic processing capabilities, including multiplication, addition, and division operations. However, for distance computation there is one non-linear function involved: $\cos^{-1}$.

3.6. Conclusions

This chapter has mainly introduced a brand-new automated approach for the unsupervised analysis of multispectral or hyperspectral image scenes that relies on mathematical morphology concepts. The extension of classic morphological operations to the hyperspectral domain enables the integration of spectral and spatial information in the analysis process and, thus, the incorporation of image processing in the field of hyperspectral analysis. In particular, a systolic array structure is proposed to speed up the performance of this new methodology, conducting an unsupervised analysis of multispectral or hyperspectral images.

The proposed methodology is applied as a rectangular structure for determining the accumulative distances, and as a linear structure of programmable processor elements to obtain the extreme values of these distances for the extraction of endmembers in the image.

The rectangular array allows us to reduce the sequential computational time of the MxMxN order to 2M+N clock cycles. This is an important advantage because of the high dimension of the used data. In the extreme distance determination phase, the supremun and infimum are simultaneously obtained.

In relation to the neural network for hyperspectral analysis, these pixels corresponding to the endmembers can be used as training patterns of

unsupervised neural networks, providing a robust and efficient solution to the abundances determination problem (hyperspectral unmixing).

Acknowledgements

This work has been conceived under the funding project *Aplicación de las imágenes hiperespectrales a la vigilancia de recursos naturales* (TIC 2000-0739-C04-3), Ministerio de Educación y Ciencia (Spain).We also wish to thank Dr. Alejandro Curado, from our university Department of English, for his linguistic revision of this chapter.

References

[1] H. T. Kung. Why Systolic Architectures?, *IEEE Trans. Computer*, pages 37-46, Jan. 1983.

[2] R. O. Greeen. *AVIRIS Earth Science Workshop Proceedings.* Available at http://makalu.jpl.nasa.gov/, 1998-2000.

[3] G. Vane, R. O. Green, T. G. Chrien, H. T. Enmark, E. G. Hansen and W. M. Porter. The Airborne Visible/Infrared Imaging Spectrometer (AVIRIS). *Remote Sensing of Environment,* 44, pages 127-143, 1993.

[4] F. A. Kruse and J. W. Boardman. Fifteen Years of Hyperspectral Data: Northern Grapevine Mountain. *Summaries of the Eighth JPL Airborne Earth Science Workshop*, Nevada,1999.

[5] D. Landgrebe,.On Progress Toward Information Extraction Methods for Hyperspectral Data, *Proc. SPIE* . San Diego, California, August-1997.

[6] F. A. Kruse. Spectral Identification of Image Endmembers Determined from AVIRIS Data. *Summaries of the Seventh JPL Airborne Earth Science Workshop*, 1998.

[7] D. A. Roberts, P. Dennison, S. Ustin, E. Reith and M. Morais. Development of a Regionally Specific Library for the Santa Monica Mountains Using High Resolution AVIRIS Data. *Summaries of the Eighth JPL Airborne Earth Science Workshop*, 1999.

[8] J. W. Boardman and F. A. Kruse. Automated Spectral Analysis: A Geological Example Using AVIRIS Data, Northern Grapevine Mountains, Nevada, *Proc. 10th Thematic Conference*, Geologic Remote Sensing, San Antonio, Texas, May 1994.

[9] F. A. Kruse and J. W. Boardman. Fifteen Years of Hyperspectral Data: Northern Grapevine Mountains. *Summaries of the Eighth JPL Airborne Earth Science Workshop*, Nevada, Pasadena, California, 1999.

[10] E. M. Winter and M. E. Winter. Autonomous Hyperspectral End-member Determination Methods. *Part of the EUROPTO Conference on Sensors,*

Systems and Next-Generation Satellites, Proc. SPIE, Florence, Italy, Sep. 1999.

[11] J. W. Boardman, F. A. Kruse and R. O. Green. Mapping Target Signatures via Partial Unmixing of AVIRIS Data. *Summaries of the Fifth JPL Airborne Earth Science Workshop*, Pasadena, California, 1995.

[12] P. Palmadesso, J. Antoniades, M. Baumback, J. Bowles and L. J. Rickard. Use of Filter Vectors and Fast Convex Set Methods in Hyperspectral Analysis. *Proceedings of SPIE*, 1999.
Available at http://nemo.nrl.navy.mil/public/publications.html.

[13] T. Szeredi, K. Staenz and R. Neville. Automated Endmembers Selection: Part I Theory. *Remote Sensing of Environment*, 1999.

[14] S. Beaven, L. E. Hoff, and E. M. Winter. Comparison of SEM and Linear Unmixing Approaches for Classification of Spectral Data. *Proc. SPIE*, 1999.

[15] L. O. Jiménez and J. Rivera-Medina, On the Integration of Spatial and Spectral Information in Unsupervised Classification for Multispectral and Hyperspectral Data. *Part of the EUROPTO Conference on Sensors, Systems and Next-Generation Satellites, Proc. SPIE*, Florence, Italy, Sep. 1999.

[16] J. Serra. *Image Analysis and Mathematical Morphology*. Academic Press, London, 1982.

[17] P. Soille. Morphological Partitioning of Multispectral Images. *Journal of Electronic Imaging*, 5,(3), pages 252-265, July 1996.

[18] P. Lambert and J. Chanussot. Extending Mathematical Morphology to Color Image Processing. *CGIP'2000*, Saint-Etienne, France, 2000.

[19] S. R. Sternberg. Greyscale Morphology. *Computer Vision Graphics and Image Processing*, 35, pages 283-305.

[20] A. Plaza, P. Martínez, J. A. Gualtieri and R. M. Pérez. Spatial/Spectral Endmember Extraction from AVIRIS Hyperspectral Data Using Mathematical Morphology. *Summaries of the JPL/AVIRIS Workshop*, Pasadena, California, 2001.

[21] R. H. Kunh. *Optimization and Interconnection Complexity for Parallel Processors, Single Stage Networks and Decision Trees*. Ph. D. Dissertation, Department of Computer Science, University of Illinois, Urbana-Champaign, 1980.

[22] D. I. Moldovan. On the Design of Algorithms for VLSI Systolic Array. *Proc. IEEE*, 71 (1), pages 113-120, 1983.