FAST IMPLEMENTATION OF HADAMARD TRANSFORM FOR
OBJECT RECOGNITION AND CLASSIFICATION USING
PARALLEL PROCESSOR

A Thesis Presented to
The Faculty of the College of Engineering and Technology
Ohio University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science

by
Saifuddin Moiz
March, 1991
CONTENTS:

Chapter 1. Introduction 1

Chapter 2. Preprocessing and Fast Hadamard Transform
2.1 Region Growing and Labelling 14
2.2 Translation and Rotation 16
2.3 Feature Extraction 24
2.3.1 Hadamard Transform 24
2.3.2 Discrete 1-D and 2-D Hadamard Transform 25
2.3.3 Fast Hadamard Transform 31
2.4 Systolic Array Architecture for FHT 41
2.5 Classification 47

Chapter 3. Development and Implementation of 2-D FHT on Parallel Processor 62
3.1 Topogy Selection 65
3.2 Memory Management of S-14 66
3.3 FHT Implementation on S-14 69
3.3 Fast Hadamard Transform Timing Analysis 72
Chapter 4. Experimental Results

4.1 Results of Histogramming and Thresholding

4.2 Results of Object Translation and Rotation

4.3 Results of Parallel Implementation of FHT

4.4 Results of Object Recognition and Classification

Chapter 5. Conclusion and Discussion

References

Appendix A

Appendix B
List of Figures

Chapter 1

Figure 1.1 General flow-graph 8

Chapter 2

Figure 2.1 Block diagram of the object recognition system 12
Figure 2.2 Noise cleaning template 13
Figure 2.3 Intensity histogram of an object 15
Figure 2.4a 6-connectivity templates 17
Figure 2.4b Result of parallel region growing and labelling algorithm 18
Figure 2.5 Orientation and centroid of the object 21
Figure 2.6 Translation and rotation of the coordinate axes 23
Figure 2.7 Natural-order Hadamard transform matrix 26
Figure 2.8 Sequency-order Hadamard transform matrix 30
Figure 2.9a Natural-order Hadamard transform of a binary image 32
Figure 2.9b Sequency-order Hadamard transform of a binary image 33
| Figure 2.10 | Computation for one-dimensional third-order Hadamard transform |
| Figure 2.11 | Hadamard transform computation sequences |
| Figure 2.12 | Computation for a particular output of Hadamard transform |
| Figure 2.13 | Storage locations for computation of one-dimensional third-order Hadamard transform |
| Figure 2.14a | Systolic array for row-wise 1-D Hadamard transform |
| Figure 2.14b | A computing cell for systolic array |
| Figure 2.15 | Column-wise 1-D Hadamard transform in the proposed systolic array to obtain 2-D Hadamard transform |
| Figure 2.16a | Result of transform coefficients normalization |
| Figure 2.16b | Result of transform coefficients normalization |
| Figure 2.17 | Two disjoint pattern classes |
| Figure 2.18a | Distance measure for one-dimensional feature vector |
| Figure 2.18b | Distance measure for two-dimensional feature vector |
| Figure 2.18c | Distance measure for three-dimensional feature vector |
| Figure 2.19a | Hadamard domain feature vector for an object |
| Figure 2.19b | Hadamard domain feature vector for an object |
| Figure 2.19a | Hadamard domain feature vector for an object |
Chapter 3

Figure 3.1a  16-node hypercube 64
Figure 3.1b  16-node ring mapped onto 16-node hypercube 64
Figure 3.2  Computation of 2-D Hadamard transform as a series of 1-D transforms 66
Figure 3.3  Image partitioning and mapping on S-14 68
Figure 3.4  A general flow-diagram of the 2-D FHT on S-14 70

Chapter 4

Figure 4.1  Experimental set-up 76
Figure 4.2  Gray-level image of an object 77
Figure 4.3  Histogram of an object 78
Figure 4.4  Binary image of an object 78
Figure 4.5  Translated image of an object 80
Figure 4.6  Rotated image of an object 80
Figure 4.7  Gray-level image of a cluster of objects 81
Tables

Table 4.1 Timing result of parallel implementation of FHT

Table 4.2 Classification results of FHT based recognition
CHAPTER 1
INTRODUCTION

Computer vision systems have found numerous applications in such areas as remote sensing, robotics, communication, and biomedical imaging, etc. In these applications, a vision system detects the objects of interest in a given picture using image processing operators (e.g., edge detection, thresholding, histogramming, region growing, split and merge, etc.). It then classifies the detected objects using the pattern recognition techniques.

Template matching [1] is one of the methods used for the detection and identification of an object in a given image. In this approach, a replica of the object of interest is compared to all unknown objects in the image field. If the match between an unknown object and the template is sufficiently close, the object is labeled as the template object. Due to inherent image noise, spatial and amplitude quantization effects, and a priori uncertainty of the exact shape and structure of an object, template matching is seldom accurate. Another major limitation of template matching is that different types of templates must often be matched against an
image field to account for changes in rotation and magnification of template objects.

Moment computations are often performed for recognition and identification purposes. Hu[2] derived a set of moment functions which have the property of invariance under image translation and rotation. Smith and Wright [3] used moments for automatic ship identification. Dudani, Breeding and McGhee[4] applied the property of invariant moments to automatic recognition of aircraft types from optical images. The major disadvantage of these methods is that they are computationally intensive.

Image transformation is another technique employed for recognizing an object in a given image. Fourier, Slant, Walsh-Hadamard, and Haar are some of the transforms used for generating pattern features. Since the linear transformation of image data results in compaction of its energy into fewer coefficients, image transformation techniques lead to more efficient and reliable classifiers. For example, in the Fourier and Walsh-Hadamard transforms the average or dc value is proportional to the average image brightness, and the high-frequency terms reveal information about the amplitude and orientation of edges within an image. As shown in [5] the Fourier descriptors are very useful for describing the shapes of closed contours. The Hadamard transform has been used for
feature selection in the character recognition task [6].

Among different techniques, the Fourier transform is distinguished by the invariance to the standard shape transformations such as scaling, rotation, and translation. However, it requires $N \log_2 N$ complex multiplications and additions (where $N$ is the number of data points to be transformed) which makes the implementation computationally expensive. On the other hand, the fast Hadamard transform utilizes the orthogonal system of Walsh functions whose elements are $+1$ and $-1$. This makes it quite appealing for the dedicated VLSI chip implementation. For $N$ number of data points, the transformation involves $N \log_2 N$ additions and/or subtractions.

Efficient and fast algorithms based on matrix partitioning and matrix factoring have further reduced the computational and memory requirements of the transforms and widened their applications [7]. For example, Carl and Swartwood [8] have designed a hybrid special purpose computer based on a recursive algorithm for the discrete Walsh–Hadamard transform computation. The device uses feedback in time to reduce the required number of summation junctions from $N \log_2 N$ to $N$. In this system, computations are completed in 700 microseconds for $N = 256$. 
Jurczyk and Loparo [9] have discussed an object recognition system which uses tactile array data and the Walsh-Hadamard transform technique. Objects are recognized according to their local features by correlating transformed data from a tactile sample with data obtained from reference images contained in a model library. Several test results are presented to illustrate the performance of the system.

Huang and Chung [10] used the Walsh functions for separating similar complex Chinese characters. They used the sequency property for the analysis of the central portion of each Chinese character. Each character is imaged 8 times by changing its size, position, and binary thresholding value. They concluded that the first 5 Walsh coefficients have most separability power to recognize each character.

It is a known fact that good classification results could be obtained by utilizing the transform coefficients for recognizing objects. Although transformation techniques have been extensively used in various recognition tasks, no attempt has been made to apply the Hadamard transform coefficients for object identification. The fast implementation of the Hadamard transform still requires lengthy computation time and excessive amounts of memory in a sequential implementation. These computations can be reduced to an acceptable level in real-time applications by using high speed image processing.
architectures and fast algorithms. In this research, parallel processing has been fully exploited to obtain the Hadamard domain features for object recognition.

In parallelizing a sequential process, problems arise when the number of processors required by the algorithms exceeds the number of processor available on the parallel system (cardinality variation) or when the communication structure of the parallel algorithm differs from the interconnection architecture of the parallel machine (topological variation). Berman and Snyder [11] have discussed several methods which produce proportionally full processor utilization and full distribution of communication paths in the algorithm over data paths in the parallel architecture. Bokhari [12] has successfully resolved the topological variation problem by creating approximation algorithms for embedding communication structures into device interconnection architectures.

Lee and Aggarwal [13] presented a mapping strategy for the parallel processing using an accurate characterization of the communication overhead. They have formulated a set of objective functions to evaluate the optimality of mapping a problem graph onto a system graph. They have also tested the mapping scheme using the hypercube as a system graph.
Kushner and Rosenfield [14] have given the general implementation of various classes of image processing tasks, such as point and local operations, transform, and statistics computation on ring, array and hypercube topologies. They have concluded that a hypercube-connected array is a promising network structure for image processing applications.

Sanz and Dinstein [15] have implemented some image transforms and features such as projections along linear patterns, convex hull approximations, Hough transform for line detection, diameter, moments, and principal components on a parallel pipeline architecture. Specifically, they presented algorithms for computing these features which are suitable for pipeline implementation of image analysis techniques.

Fang, Li and Ni [16] have presented several parallel algorithms for image template matching on an SIMD (single instruction multiple data) array processor with a hypercube interconnection network. With efficient use of the inter-PE communication network, each PE (processing element) requires only a small local memory, and the time complexity is greatly reduced.

Mudge and Abdel-Rahman [17] have developed a general model for the hypercube machines to show the execution of vision algorithms on these systems. A thick film circuit
inspection task is implemented to evaluate the hypercube machine performance. Based on their study, they concluded that hypercube machines have great potential for low and intermediate level vision algorithms.

In this research, the fast Hadamard transform (FHT) has been implemented on a hypercube concurrent processor (Ametek S-14) by exploiting its network-embedding feature. The SIMD architecture of the S-14 is configured in ring topology to achieve maximum computational efficiency with minimum internode communication.

A flow graph of the overall approach is shown in Figure 1.1. The two phases involved in object recognition are training and classification. In order to train the classifier, individual objects are exposed to the camera and fast Hadamard transform algorithm is applied to extract Hadamard domain features. This process starts by histogramming and thresholding the input image to obtain the binary picture. A 3x3 local window is applied to each pixel to remove noise from a given image. To make the classification invariant to position and orientation, the object is first translated to the center of the image plane by a coordinate axis transformation. Second, axes of least moment of inertia are calculated and the test image is rotated with respect to the pattern image. The preprocessed image is divided among 16-
Present each object to the system and obtain an ascii image

Obtain a binary image of the object by histogramming and thresholding

Translate the binary image to the center of image plane and align it with the objects principal axes for position invariant classification

Apply 2-D fast Hadamard transform to the binary image and normalize the resultant Hadamard domain coefficients

Extract the significant Hadamard domain features of the object

Is database complete?

No

Get a new image and repeat till the database is complete

Yes

Present a cluster of objects, preprocess and isolate using a parallel region growing and labelling algorithm

Classify the given objects using a minimum distance criterion

Stop

Figure 1.1 General flow-graph
nodes of the concurrent system with each node receiving a 32x512 subimage. Using the separability property of the two-dimensional (2-D) Hadamard kernel, the 2-D Hadamard transform of an image is obtained by two successive applications of an one-dimensional (1-D) Hadamard transform algorithm to the subimage in each node of the parallel processor as follows. A 1-D row transform algorithm is applied to each subimage simultaneously. The subimages are rearranged and a 1-D column transform is applied to obtain a 2-D FHT of the subimage. Subimages from processor nodes are uploaded to the parent system (VAX 11/750) to get the 2-D transformed image. To make the classifier independent of size, the Hadamard domain coefficients are normalized by dividing each coefficient by the element (0,0) of the transformed image. A 1-D reference feature vector for each object is obtained from this matrix by an adaptive algorithm which uses intraclass distance and interclass distance as the optimization criterion.

During the classification phase, a cluster of objects is presented to the system for identification. The objects are isolated using a parallel region growing and labelling algorithm. Each object is preprocessed and transformed into the Hadamard domain to obtain a feature vector. The minimum distance rule is used to determine which reference feature vector best matches the unknown test feature vector. According to this rule, an unknown object represented by a feature
vector $x$ belongs to class $w_i$ if

$$d_i(x) > d_j(x), \quad j = 1,2,3,4 \ldots M-1, \ j \neq i,$$

where $M$ is the number of classes and $d_k(x)$ is the Euclidean distance between the test feature vector $x$ and the $k$th reference feature vector.

Implementing the Hadamard transform technique on a parallel processor resulted in a considerable speedup. The achieved speedup is measured for different parallel algorithms developed in this research and compared with the theoretical values.

The remainder of this thesis is organized as follows. In Chapter 2, preprocessing techniques are described first, then a detailed review of the fast the Hadamard transform and the design of the classifier are explained. Chapter 3 discusses the parallel implementation of the Hadamard transform on the S-14, and provides the theoretical speedup analysis. In Chapter 4, we present experimental results and the speedup achieved for these implementations. Finally Chapter 5 provides the conclusions and the further research topics.
The vision system devised in this research consists of two modules: image acquisition unit and processing system. The image acquisition system obtains an 8-bit gray-level image of the given object through an RCA 2000 video camera and a VICOM image processor, and stores it in an array of 512x512 pixels. The processing system consists of a preprocessor which removes the inherent noise due to the quantization of the sampling system and the improper lighting conditions. After preprocessing, the Hadamard transformation technique is applied to obtain a set of feature vectors for different objects. These features are used for object recognition. The above process is shown in Figure 2.1 and explained in detail in the following sections.

The process starts by removing the inherent noise in the given digital image represented as \( \{ f(x,y) : x = 0,1,\ldots,M-1 \text{ and } y = 0,1,\ldots,N-1 \} \) where \( f(x,y) \) is the gray-level of each pixel varying between 0 and 255. This is achieved by applying a 3x3 local averaging window as shown in Figure 2.2. The average gray-level of the central pixel \( g(x,y) \) is calculated
Patterns or objects to be recognized

Image acquisition using VICOM

Preprocessing
(smoothing, histogramming, thresholding)

Translate and rotate the binary image

Feature extraction
( Hadamard domain coefficients )

Classification
( minimum-distance criterion )

Figure 2.1 Block diagram of the object recognition system
<table>
<thead>
<tr>
<th>G1</th>
<th>G2</th>
<th>G3</th>
</tr>
</thead>
<tbody>
<tr>
<td>G4</td>
<td>G5</td>
<td>G6</td>
</tr>
<tr>
<td>G7</td>
<td>G8</td>
<td>G9</td>
</tr>
</tbody>
</table>

Figure 2.2 Noise cleaning template
by using the following equation,

\[ g(x,y) = \frac{(G1+G2+G3+G4+G5+G6+G7+G8+G9)}{9} \]  

(2.1)

where G1, G2, ..., G9 are the gray-level values of the pixels in the 3x3 window. To obtain a binary image, we form the gray-level histogram as shown in Figure 2.3. A threshold value is chosen by defining the following criterion,

\[ h(x,y) = 1 \quad \text{if } f(x,y) \geq T \]
\[ h(x,y) = 0 \quad \text{otherwise} \]

where T is the selected threshold. When the scene consists of more than one object, a unique label is assigned to each object by using region growing and labelling technique.

### 2.1 Region Growing and Labelling

The process of assigning labels to objects in a discrete binary image is to choose a point where \( f(x,y) = 1 \) and assign a label to this point and its neighbors. Next, we label all the neighbors of these neighbors (except those that have already been labeled), and so on. When this recursive procedure terminates one component will have been labeled completely. Then we continue by choosing another point. This assignment process is called connected component labeling.

A region growing and labelling algorithm described in [18] is used in this research. The method starts by scanning
Figure 2.3 Intensity histogram of an object
the image from the top left hand corner. Once an object pixel is detected, the scanning is stopped and the pixel is accepted as a seed point. The detected seed is assigned a unique gray-level value, which is taken as the starting position of a region. The region growing continues by appending to the seed point its 6-connected pixels as shown in Figure 2.4a (where P is the central pixel and x's are the neighboring pixels). Each connected pixel receives the label of the seed point which in turn becomes a new seed point. In this way, an image region is formed by pixel aggregation. This process is repeated as many times as the number of objects present in the scene. It terminates when no more seed points are available. The result of this algorithm is shown in Figure 2.4b. Once the objects are assigned unique labels, each object is translated and rotated to make the classification independent of position and orientation.

2.2 Translation and Rotation.

The Hadamard transform domain coefficients are dependent on the position and orientation of the object. This implies that the classifier will render accurate results under the assumption that the objects are oriented and positioned in the same way as the prototypes. To make the classifier invariant to above mentioned changes, a coordinate transformation technique is employed as explained below.
Figure 2.4a 6-connectivity templates
Figure 2.4b Result of parallel region growing and labelling
The area of the binary image of the given object is computed (in units of the area of picture cell) by using

\[
\text{Area} = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} b_{ij}
\]  

(2.2)

where \(b_{ij}\) is the binary image value at the point in the \(i\)th row and \(j\)th column, and the summation is taken over the entire image. To determine the position of the object we need the center of the object, which is defined as the center of mass of a figure of the same shape with constant mass per unit area. The center of mass is a point where all the mass of the object could be concentrated without changing the first moment of the object about any axis. For a two-dimensional case, the coordinates \((\bar{x}, \bar{y})\) of the object's center are given by

\[
\bar{x} = \frac{\sum_{i=0}^{N-1} \sum_{j=0}^{N-1} ib_{ij}}{\sum_{i=0}^{N-1} \sum_{j=0}^{N-1} b_{ij}}
\]  

(2.3)

\[
\bar{y} = \frac{\sum_{i=0}^{N-1} \sum_{j=0}^{N-1} jb_{ij}}{\sum_{i=0}^{N-1} \sum_{j=0}^{N-1} b_{ij}}
\]  

(2.4)

The second moments about the \(x\) and \(y\) axis are defined as

\[
a = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} (x_i - \bar{x})^2 b_{ij}
\]  

(2.5)
The diagonal moment is defined as

\[ b = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} (y_j - \bar{y})^2 b_{ij} \]  

(2.6)

The diagonal moment is defined as

\[ c = 2 \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} (x_i - \bar{x})(y_j - \bar{y})b_{ij} \]  

(2.7)

After calculating the above moments, the orientation of an object is calculated as the direction of the axis of least inertia which in turn is the axis about which the second moment of a sheet of material is the smallest. Let \( \theta \) be the angle of the orientation defined as the angle made by axis of least inertia with the x-axis and measured in a counterclockwise direction. It is given by

\[ \theta = \frac{1}{2} \tan^{-1}(b/(a-c)) \]  

(2.8)

The orientation and centroid of the object are shown in Figure 2.5. The sign of the diagonal moment \( b \) gives the direction of orientation of the axis of least inertia with respect to the x-axis. A negative sign indicates an angle of orientation in the clockwise direction and vice versa.

Once the orientation of the object is determined, the image coordinates axes are aligned with the principal axes of
Figure 2.5 Orientation and centroid of the given object
the object. To obtain the transformation, we take the centroid of the object as the new origin of the coordinate system and represent each pixel location \((x,y)\) by its polar coordinates as

\[ x = r \cos \alpha, \quad y = r \sin \alpha \]

Here \(r\) is the distance between the object pixel \((x,y)\) and the object centroid \((\bar{x},\bar{y})\) and \(\alpha\) is the orientation of each pixel with respect to the \(x\)-axis. Let the new coordinates be expressed as \((x'',y'')\). Then from Figure 2.6

\[ x' = x - \bar{x} \quad \text{or} \quad x = x' + \bar{x} \tag{2.9} \]
\[ y' = y - \bar{y} \quad \text{or} \quad y = y' + \bar{y} \tag{2.10} \]

and

\[ x'' = r \cos (\alpha - \theta) \]
\[ = r \cos \alpha \cos \theta + r \sin \alpha \sin \theta \]

Since \(x' = r \cos \alpha\) and \(y' = r \sin \alpha\), we have

\[ x'' = x' \cos \theta + y' \sin \theta \tag{2.11} \]

and

\[ y'' = -x' \sin \theta + y' \cos \theta \tag{2.12} \]

Once this rotation and translation is completed, the Hadamard transform is applied to obtain a feature vector for each object.
Figure 2.6 Translation and rotation of the coordinate axes
2.3 Feature Extraction

This section deals with the development of the 2-D discrete Hadamard transform for feature extraction. An efficient algorithm is implemented for computing the fast Hadamard transform. Timing analysis is provided for both serial and parallel realizations.

2.3.1 Hadamard Transform

The Hadamard transform has been used in communication for image transmission and in pattern recognition for feature extraction [19]. The Hadamard transform consists of orthogonal Walsh functions. Each row (or column) of the Hadamard matrix is a Walsh function defined in the interval (-1/2, 1/2). The elements of a Hadamard matrix take the values of +1 or -1.

The Hadamard transform image coding technique has been shown to provide good quality image transmission with the same number of bits as required to code the spatial domain of an image by conventional pulse code modulation. Furthermore, transmission of the Hadamard domain image rather than the image itself offers a certain immunity to channel errors and the ability to achieve a bandwidth reduction. Bandwidth reduction is possible because the image energy which is uniformly distributed in the spatial domain, tends to be concentrated near the origin of the Hadamard domain. Many of the higher spatial frequency components are of very low
magnitude, and need not to be transmitted. Boules [20] has applied the Walsh-Hadamard transformation for the fast implementation of the least-mean-square error (LMS) adaptive transversal filter. This filter gives a significant reduction in computation over both the conventional time domain and the frequency-domain LMS adaptive filter.

### 2.3.2 Discrete 1-D and 2-D Hadamard transform

If $H$ is an $N \times N$ Hadamard matrix then the product of $H$ and its transpose is the $N \times N$ identity matrix $I$. Thus

$$(1/N) \ H \ H^T = I \quad (2.13)$$

If $H$ is a symmetric Hadamard matrix then 2.13 reduces to

$$(1/N) \ H \ H = I \quad (2.14)$$

The lowest-order Hadamard matrix is of order two,

$$H(2) = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

Hadamard matrices of higher order can be generated by using the recursive property of the Hadamard matrix that is

$$H(2N) = \begin{bmatrix} H(N) & H(N) \\ H(N) & -H(N) \end{bmatrix}$$

where $H(N)$ represents the matrix of order $N$ and $H(2N)$ is the Hadamard matrix of order $2N$ assuming $N = 2^n$. A natural order Hadamard matrix for $N = 8$ is shown in Figure 2.7. Let the array $f(x)$ represent the intensity samples of an original image over an array of $N$ points. Then the one-dimensional Hadamard transform $F(u)$, of $f(x)$, is given by the matrix
Figure 2.7 Natural-order Hadamard matrix

\[
\begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & -1 & +1 & -1 & +1 & -1 & +1 & -1 \\
1 & +1 & -1 & -1 & +1 & +1 & -1 & -1 \\
1 & -1 & -1 & +1 & +1 & -1 & -1 & +1 \\
1 & +1 & +1 & +1 & -1 & -1 & -1 & -1 \\
1 & -1 & +1 & -1 & -1 & +1 & -1 & +1 \\
1 & +1 & -1 & -1 & -1 & +1 & +1 & +1 \\
1 & -1 & -1 & +1 & -1 & +1 & +1 & -1 \\
\end{bmatrix}
\]
product

\[ [F(u)] = [H(u)][f(x)][H(u)] \]

where \([H(u)]\) is a symmetric Hadamard matrix of order \(N\). The one-dimensional forward Hadamard transform of a discrete sequence \(f(x)\) can be written in series form as

\[
H(u) = \frac{1}{N} \sum_{x=0}^{N-1} f(x) (-1)^{\sum_{i=0}^{n-1} b_i(x) b_i(u)}
\tag{2.15}
\]

for \(u = 0, 1, 2, \ldots, N-1\), where \(N = 2^n\) and \(b_z(k)\) is the \(k\)th bit in the binary representation of \(z\). For example, if \(n = 3\) and \(z = 6\) (110 in binary), we have \(b_0(z) = 0\), \(b_1(z) = 1\), and \(b_2(z) = 1\).

The inverse Hadamard transform is equal to the forward Hadamard transform multiplied by \(N\), which is given by

\[
f(x) = \sum_{u=0}^{N-1} H(u) (-1)^{\sum_{i=0}^{n-1} b_i(x) b_i(u)}
\tag{2.16}
\]

for \(x = 0, 1, 2, \ldots, N-1\). The two-dimensional natural order Hadamard transform pair is given as

\[
H(u,v) = \frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y) (-1)^{\sum_{i=0}^{n-1} [b_i(x)b_i(u) + b_i(y)b_i(v)]}
\tag{2.17}
\]

for \(u = 0, 1, 2, \ldots, M-1\), and \(v = 0, 1, 2, \ldots, N-1\), and
for \( x = 0, 1, 2, \ldots, M-1 \), and \( y = 0, 1, 2, \ldots, N-1 \)

For \( NxN \) square images the above transform pair is reduced to

\[
H(u, v) = \left( \frac{1}{N} \right) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x, y) (-1)^i \sum_{i=0}^{n-1} [b_i(x)b_i(y) + b_i(y)b_i(v)] \quad (2.19)
\]

for \( u, v = 0, 1, 2, \ldots, N-1 \)

Since the two-dimensional Hadamard kernels are separable, the 2-D transform pair can be obtained by successive applications of the one-dimensional Hadamard transform algorithm as follows.

\[
H(u, v) = \left( \frac{1}{\sqrt{N}} \right) \sum_{x=0}^{N-1} (-1)^i \sum_{i=0}^{n-1} b_i(x)b_i(u) \quad (1/\sqrt{N}) \sum_{y=0}^{N-1} f(x, y) (-1)^i \sum_{i=0}^{n-1} b_i(y)b_i(v) \quad (2.21)
\]

for \( u, v = 0, 1, 2, \ldots, N-1 \)

The number of sign changes along a column (row) of the Hadamard matrix is called the sequency of that column (row). Hadamard kernels are expressed so that the sequency increases as a function of increasing \( u \). This is analogous to the
frequency concept of the Fourier transform in the sense that frequency also increases as a function of increasing $u$. The 1-D sequency ordered transform pair is given by

$$H(u) = \frac{1}{N} \sum_{x=0}^{N-1} f(x) (-1)^{i\cdot u}$$ (2.22)$$

$$f(x) = \sum_{u=0}^{N-1} H(u) (-1)^{i\cdot u}$$ (2.23)

where $p_0(u) = b_{n-1}(u)$,

\[ \ldots \]

\[ p_{n-1}(u) = b_1(u) + b_0(u) \]

for $x, u = 0, 1, 2, \ldots, N-1$

The 2-D Hadamard sequency ordered transform pair is given by

$$H(u, v) = \frac{1}{N} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x, y) (-1)^{i\cdot u}$$ (2.24)

for $u, v = 0, 1, 2, \ldots, N-1$ and

$$f(x, y) = \frac{1}{N} \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} H(u, v) (-1)^{i\cdot u}$$ (2.25)

for $x, y = 0, 1, 2, \ldots, N-1$. An example of Hadamard matrix in its sequency ordered form is shown in Figure 2.8.
Figure 2.8 Sequency-order Hadamard matrix

\[
\begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & +1 & +1 & +1 & -1 & -1 & -1 & -1 \\
1 & +1 & -1 & -1 & -1 & +1 & +1 & +1 \\
1 & +1 & -1 & -1 & +1 & +1 & -1 & -1 \\
1 & -1 & -1 & +1 & +1 & -1 & -1 & +1 \\
1 & -1 & -1 & +1 & -1 & +1 & +1 & -1 \\
1 & -1 & +1 & -1 & -1 & +1 & -1 & +1 \\
1 & +1 & -1 & +1 & -1 & +1 & -1 & +1 \\
\end{bmatrix}
\]
The two-dimensional Hadamard transform may be computed in either natural or ordered form with an algorithm analogous to the fast Fourier transform algorithm. The natural order transform is more suitable for applications involving a double transform (time-space-time) such as logical autocorrelation and convolution, whereas a sequency-ordered set of output coefficients are useful in applications such as sequency filters and sequency power spectra etc. To determine which order is more suitable for this research, we performed a number of experiments. Figure 2.9a shows results of one such experiment. As seen from the Figure 2.9b the coefficients of the sequency-ordered transform seem to be in a particular order and as the sequency increases, the magnitude of the samples tend to decrease. This indicates that there are relatively few high-amplitude brightness transitions between elements in the original image. Based on these observations the sequency-ordered Hadamard transform is chosen for this research.

2.3.3 Fast Hadamard Transform

The computation of equation 2.24 is performed in two steps. First a one-dimensional Hadamard transform is taken along each row of the array \( f(x,y) \), which yields

\[
H(u,y) = \frac{1}{N} \sum_{x=0}^{n-1} f(x,y) (-1)^{x} \sum_{i=0}^{n-1} \left[ b_{1}(x)p_{1}(u) \right] 
\]  

(2.26)

Then a second one-dimensional Hadamard transform is taken
Binary image

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

The Natural-order Hadamard transform

52 0 0 20 0 20 20 0 0 -28 -28 0 -28 0 0 -28
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
-20 0 0 12 0 12 12 0 0 -4 -4 0 -4 0 0 -4
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
-20 0 0 12 0 12 12 0 0 -4 -4 0 -4 0 0 -4
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
20 0 0 -12 0 -12 -12 0 0 4 4 0 4 0 0 4
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
28 0 0 -4 0 -4 -4 0 0 -4 -4 0 -4 0 0 -4
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
-28 0 0 4 0 4 4 0 0 4 4 0 4 0 0 4
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
-28 0 0 4 0 4 4 0 0 4 4 0 4 0 0 4
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
28 0 0 -4 0 -4 -4 0 0 -4 -4 0 -4 0 0 -4
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 2.9a Natural-order Hadamard transform of a binary image
Binary image

\[
\begin{array}{cccccccccccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

The sequency-order Hadamard transform

\[
\begin{array}{cccccccccccccccc}
52 & 0 & -28 & 0 & 0 & -28 & 20 & 0 & 20 & 0 & 0 & -28 & 0 & -28 & 20 & 0 \\
28 & 0 & 0 & -4 & 0 & -4 & -4 & 0 & 0 & -4 & -4 & 0 & -4 & 0 & 0 & -4 \\
-28 & 0 & 0 & 4 & 0 & 4 & 4 & 0 & 0 & 4 & 4 & 0 & 4 & 0 & 0 & 4 \\
-20 & 0 & 0 & -4 & 0 & -4 & 12 & 0 & 0 & -4 & 12 & 0 & 12 & 0 & 0 & -4 \\
-20 & 0 & 0 & -4 & 0 & -4 & 12 & 0 & 0 & -4 & 12 & 0 & 12 & 0 & 0 & -4 \\
-28 & 0 & 0 & 4 & 0 & 4 & 4 & 0 & 0 & 4 & 4 & 0 & 4 & 0 & 0 & 4 \\
20 & 0 & 0 & 4 & 0 & 4 & -12 & 0 & 0 & 4 & -12 & 0 & -12 & 0 & 0 & 4 \\
28 & 0 & 0 & -4 & 0 & -4 & -4 & 0 & 0 & -4 & -4 & 0 & -4 & 0 & 0 & -4 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}
\]

Figure 2.9b Sequency-order Hadamard transform of a binary image
along each column of $H(u, y)$ giving the desired result,

$$H(u, v) = \sum_{y=0}^{n-1} H(u, y)(-1)^{b_1(y)p_1(u)}$$  \hspace{1cm} (2.27)

Various fast computational algorithms for the Hadamard transform have been reported. Since the Hadamard transform contains only ±1 values, no multiplications are required in the transform calculations. Therefore, the number of additions or subtractions required can be reduced from $N^2$ to $N \log_2 N$. This comes from the fact that Hadamard matrix can be written as a product of $n$ sparse matrices, that is,

$$\hat{H} = \hat{H}_n = H^n, \quad n = \log_2 N$$

where

$$\hat{H} \triangleq 1/\sqrt{2} \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & & \cdots & 1 & 1 \\ -1 \end{bmatrix}$$

Since $\hat{H}$ contains only two nonzero terms per row, the transformation

$$v = (\hat{H}_n^n)u = \hat{H} \hat{H} \cdots \hat{H}u, \quad n = \log_2 N$$

can be accomplished by operating $\hat{H}$ $n$ times on $u$. Due to the structure of $\hat{H}$ only $N$ additions or subtractions are required each time $\hat{H}$ operates on a vector, giving a total of $N \times n = N$
\log_2 N additions or subtractions. Good [21] described a matrix
decomposition technique which can be implemented to perform
the Hadamard transform with \(N \log_2 N\) operations. This is similar
to the fast Fourier transform algorithm, where computational
savings are obtained by storage of intermediate results.

The algorithm described in [22] provides the transformed
coefficients in "ordered" form which correspond to the order
of increasing sequency. However, it requires \(N/2\) auxiliary
storage locations where \(N = 2^n\) is the number of input samples.
Shanks [23] proposed a method which performs the computation
"in place" meaning that the intermediate results are stored in
the location of the input samples without any need for
auxiliary storage. However, the transformed coefficients are
in the "natural" form but in the order of increasing sequency.
The computation algorithm for the Hadamard transform in
ordered form is given by Ulman [24]. This method requires \(N\)
auxiliary storage locations for intermediate results. The
methods used in this research is described in Figure 2.10. It
illustrates the computation of one-dimensional Hadamard
transformation for eight data points. The data points are
arranged in a column at level 3 and then summed by pairs to
produce intermediate results for level 2. A dotted line
linking two nodes indicates that the data points at the higher
level are multiplied by minus one before addition, or
equivalently, the data points form the subtrahend of a
Figure 2.10 Computation for one-dimensional third-order Hadamard transform
subtraction operation. Operations follow the tree graph to level 0 which is the ordered Hadamard transform of \( f(x) \). There are two operations performed at each node of levels 1, 2, and 3 yielding a total of \( 8 \log_2 8 = 24 \) operations.

The fast Hadamard transform algorithm performs the operations indicated in Figure 2.10 in a certain selected order. All operations at level \( k \) are not completed before proceeding to level \( k-1 \), but rather operations are performed according to a sieving sequence. Figure 2.11 describes the basic sequence computations. The first sequence is the "5" sequence in which the sum of all data points is formed to produce \( F(0) \). A "1" sequence subtracts the lower node from the upper node of a pair of nodes at level \( k-1 \) to produce a result at level \( k \). In the "2" sequence, operations begin at level \( k-2 \) where pairs are subtracted from one another to produce the results of level \( (k-1) \) which in turn are added together. The "3" sequence and higher sequences to the "n" sequence follow directly. Figure 2.12 shows the specific coefficients required to obtain a particular Hadamard transform output.

Original data are stored in a block of \( N \) words corresponding to level \( n \). Intermediate results are stored in \( (n-1) \) different blocks of sizes varying from \( 2^{n-1} \) to \( 2^2 \) words. These storage locations correspond to levels from \( (n-1) \) to 1 in the computation procedure. Figure 2.13 gives the storage
Figure 2.11 Hadamard transform computational sequences
Figure 2.12 Computation for a particular output of Hadamard transform
Figure 2.13 Storage locations for computation of one-dimensional third order Hadamard transform
locations for computation of a one-dimensional Hadamard transform for \( N = 8 \). Characteristic feature vectors of individual objects are extracted from Hadamard domain coefficients by utilizing the procedure described in section 2.5. The Hadamard transform involves addition or subtraction operations, which makes it quite appealing for a VLSI implementation as explained in the following section.

2.4 Systolic Array Architecture for FHT

We propose a systolic array architecture for fast and efficient implementation of the Hadamard transform. It consists of a linear array of processing elements (cells) connected in a two-dimensional mesh configuration as shown in Figure 2.14a. Each cell executes the following multiplication and addition operation (see Figure 2.14b) on its operands

\[
[H] = [H'] + [A][B] \tag{2.28}
\]

and

\[
AxB = \sum_{k=0}^{n} a_{ik} x b_{kj}, \forall (i,j) \tag{2.29}
\]

where \( H \) is the 1-D Hadamard transform coefficients matrix, \( H' \) is the partial sum of the transform coefficients, \( A \) is the Hadamard matrix of order \( NxN \), \( B \) is a matrix of order \( NxN \) whose transform is required, and \([a_{ij}]\), \([b_{ij}]\) represent the element in the ith row and jth column of \( A \) and \( B \) matrices respectively.
Figure 2.14a Systolic array for row-wise 1-D Hadamard transform
Figure 2.14b A computing cell for systolic array

\[ H(t-1) = a(t)b(t) + H'(t) \]
Each cell $C_{ij}$ is designed to receive its operands (a and b) from the left and top cells respectively. In addition to computing $H$, $C_{ij}$ propagates its preceding a and b input operands rightward and downward, respectively. The N operands forming the ith row of (A) flow horizontally from left to right through the ith row of cells, and N operands forming the jth column of (B) flow vertically through the jth column of cells in a similar fashion. The a and b operands are carefully ordered and separated by zeros as shown in Figure 2.14 so that the specific operands pairs $a_{ik}$ and $b_{kj}$ appearing in equation 2.29 meet at an appropriate cell of the array. Here they are multiplied according to equation 2.29, and added to a running sum $H'$. The $H'$s are emitted from the left side of $C_{ij}$ so that there is a flow of partial results from right to left through the array. Each row of cells eventually issues the corresponding row of the 1-D transform coefficients from its left side.

To show the operation of the proposed architecture, we consider the case for $N = 3$. In this case equation 2.28 becomes

$$H_{11} = a_{11} b_{11} + a_{12} b_{21} + a_{13} b_{31}$$

The operand $a_{11}$ flows rightward through the top row of cells meeting only zero values of a and b, until it encounters $b_{11}$ at cell $C_{13}$ at time $t = 3$. This cell then computes $H = a_{11} b_{11} + 0$ and forwards it to the cell $C_{12}$ located at its left. At the
same time, $C_{13}$ forwards $b_{11}$ to the second row of cells for computing the second row of the resultant matrix $H$. It also forwards $a_{11}$ to its right neighbor $C_{14}$. In the next clock cycle ($t=4$), $a_{12}$ and $b_{21}$ are applied to cell $C_{12}$. This cell therefore computes $H = a_{12}b_{21} + H'$, where $H' = a_{11}b_{11}$. Finally, at $t=5$ the last pair of operands $a_{13}$ and $b_{31}$ converge at the boundary cell $C_{11}$, which computes $H = a_{13}b_{31} + H'$ using the value $H' = a_{11}b_{11} + a_{12}b_{21}$ determined by $C_{12}$ and giving the desired result $H_{11}$. At time $t=6$ $C_{11}$ generates a zero output, and $t=7$ it produces the next element $H_{12}$ of $H$. This process continues until all the elements of the first row of transform coefficients have been generated. Simultaneously, the remaining rows of cells compute the other rows of $H$.

The described systolic array takes the bit-map of the binary image as input and computes the Hadamard transform. The transform of the rows is performed simultaneously in $O(n)$ steps. The pixel values are shifted so that each cell receives all the values in a particular row. It then multiplies each one by the appropriate weighing factor (+1 or -1) and adds it to a cumulative sum so that when they have all been accumulated the node computes its output value. Next the image is transposed by cyclically shifting the data with each node keeping those pixels belonging to a given column. This takes only $O(n)$ steps. Finally, the nodes simultaneously compute the column transform as shown in Figure 2.15 which takes $O(n)$
Figure 2.15 Column-wise 1-D Hadamard transform in the proposed systolic array to obtain 2-D Hadamard transform.
steps. This gives the total computation time in the magnitude order of \( O(n \log_2 n) \).

### 2.5 Classification

One of the properties of the Hadamard transform is that the zero sequency term,

\[
F(0,0) = \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x,y) \tag{2.30}
\]

is a measure of the average brightness of an object. If \( f(x,y) \) is a positive real function, then the maximum possible value of the zero sequency term is \( N^2(A) \) where \( A \) is the maximum value of \( f(x,y) \). All Hadamard domain samples other than the zero sequency sample range between \( \pm N^2(A/2) \). The magnitude of the zero sequency term is a bound for the magnitude of all other Hadamard domain samples. Since the Hadamard domain coefficients change with the size of the object, normalization is achieved by dividing each coefficient by the zero sequency element of the transform matrix. This results in transform coefficients invariant of the object size as shown in Figure 2.16a, and Figure 2.16b.

The Hadamard domain feature vector contains the transform coefficients which are real numbers. Thus a pattern or feature vector is a point in an n-dimensional Euclidean space. The set of patterns belonging to the same class corresponds to an ensemble of points scattered within some region of measurement.
The original image of an object

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

The normalized image

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
0 1.0 1.0 1.0 1.0 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5
1 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5
2 1.0 1.0 1.0 1.0 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5
3 1.0 1.0 1.0 1.0 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5
4 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5
5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5
6 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5
7 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5
8 0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3
9 0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3
10 0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3
11 0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3
12 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5
13 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5
14 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5
15 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5

Figure 2.16a Result of transform coefficients normalization
Twice the size of original image of an object

| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 |
| 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 |
| 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 |
| 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 |
| 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 |
| 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 |
| 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 |
| 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 |
| 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 |
| 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 |
| 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 |

The normalized image

| 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 |
| 0 1.0 1.0 1.0 1.0 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 |
| 1 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 |
| 2 1.0 1.0 1.0 1.0 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 |
| 3 1.0 1.0 1.0 1.0 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 |
| 4 0.5 0.5 0.5 0.5 0.5 0.7 0.7 0.7 0.7 0.7 0.7 0.7 0.7 0.7 |
| 5 0.5 0.5 0.5 0.5 0.5 0.7 0.7 0.7 0.7 0.7 0.7 0.7 0.7 0.7 |
| 6 0.5 0.5 0.5 0.5 0.5 0.7 0.7 0.7 0.7 0.7 0.7 0.7 0.7 0.7 |
| 7 0.5 0.5 0.5 0.5 0.5 0.7 0.7 0.7 0.7 0.7 0.7 0.7 0.7 0.7 |
| 8 0.3 0.3 0.3 0.3 0.3 0.7 0.7 0.7 0.7 0.7 0.7 0.7 0.7 0.7 |
| 9 0.3 0.3 0.3 0.3 0.3 0.7 0.7 0.7 0.7 0.7 0.7 0.7 0.7 0.7 |
| 10 0.3 0.3 0.3 0.3 0.3 0.7 0.7 0.7 0.7 0.7 0.7 0.7 0.7 0.7 |
| 11 0.3 0.3 0.3 0.3 0.3 0.7 0.7 0.7 0.7 0.7 0.7 0.7 0.7 0.7 |
| 12 0.5 0.5 0.5 0.5 0.5 0.7 0.7 0.7 0.7 0.7 0.7 0.7 0.7 0.7 |
| 13 0.5 0.5 0.5 0.5 0.5 0.7 0.7 0.7 0.7 0.7 0.7 0.7 0.7 0.7 |
| 14 0.5 0.5 0.5 0.5 0.5 0.7 0.7 0.7 0.7 0.7 0.7 0.7 0.7 0.7 |
| 15 0.5 0.5 0.5 0.5 0.5 0.7 0.7 0.7 0.7 0.7 0.7 0.7 0.7 0.7 |

Figure 2.16b Result of transform coefficients normalization
space. A simple example is shown in Figure 2.17 for pattern
classes \( w_i \) and \( w_j \). The pattern vectors are of the form \( X^i = (x_1^i, x_2^i)^T \), where \( x_1^i \) and \( x_2^i \) represent different Hadamard domain
coefficients and \( Z_i \) and \( Z_j \) are the cluster center for these
classes. In \( n \)-dimensional pattern space, the mean-square
intraset distance \( d_{ii} \) and the mean-square interset distance \( D_{ij} \)
are calculated as

\[
\begin{align*}
    d_{ii}^2(x^i, x^i) &= \frac{1}{K(K-1)} \sum_{j=1}^{K} \sum_{i=1}^{K} \sum_{k=1}^{n} (x_k^i-x_k^i)^2 \quad i,j=1,2,\ldots,K-1; i \neq j \\
    D_{ij}^2(x^i, y^j) &= \frac{1}{K(K-1)} \sum_{j=1}^{K} \sum_{i=1}^{K} \sum_{k=1}^{n} (x_k^i-y_k^j)^2 \quad i,j=1,2,\ldots,K-1; i \neq j
\end{align*}
\]

where \( K \) represent number of samples in a pattern class, and \( n \)
is the number of coefficients in the feature vector. These
distance measures are used to select a set of features which
are best representative of a particular class of objects. Let
\( H(m,n) \) represent the coefficients of Hadamard transform
matrix, where \( m,n = 0,1,\ldots,N-1 \). We select Hadamard domain
coefficients \( H(m,n) \) subject to the constraints that

\[
    D_{ij} \bigg|_{\text{max}} \quad \text{and} \quad \quad d_{ii} \bigg|_{\text{min}}
\]

This is equivalent to solving the following two equations in
order to find the maximum and minimum of the above mentioned
Figure 2.17 Two disjoint pattern classes
distance measures.

\[ \frac{\partial d_{ii}}{\partial H(m,n)} = 0 \ \forall (m,n) \]

\[ \frac{\partial D_{ij}}{\partial H(m,n)} = 0 \ \forall (m,n) \]

where \( m,n = 0,1, \ldots \ldots ,N-1 \).

Since solving these equations mathematically is difficult, we provide an optimization process to obtain the desired feature coefficients. The process starts by taking the \( H(0,1) \) element of class \( w_i \) and \( w_j \), and calculates the distances \( d_{ii}(1) \), \( d_{jj}(1) \), and \( D_{ij}(1) \) as shown in Figure 2.18a. Then it includes the Hadamard coefficient \( H(0,2) \) to make the feature space two-dimensional, and calculates the distances \( d_{ii}(2) \), \( d_{jj}(2) \), and \( D_{ij}(2) \) as shown in Figure 2.18b. We compare the results of the above set of distances and if

\[ d_{ii}(2) < d_{ii}(1) \]
\[ d_{jj}(2) < d_{jj}(1) \]
\& \[ D_{ij}(2) > d_{ij}(1) \]

we include one more element to the feature vector (see Figure 2.18c.) and do the above calculations, otherwise we stop and take these elements as the best representative of that particular class of object. If we let \( X^i = (x_1^i, x_2^i, x_3^i, \ldots \ldots , x_k^i)^T \) to represent the feature vector of a particular object at the kth iteration step, the above
Figure 2.18 Distance measure for 1-D, 2-D and 3-D feature space
process can be described as if

\[ |d_{ij}(k)| < |d_{ij}(k-1)| \]

\[ |d_{jj}(k)| < |d_{jj}(k-1)| \]

\[ |D_{ij}(k)| > |D_{ij}(k-1)| \forall (i,j) \]

Then \( X^i = (x_1^i, x_2^i, x_3^i, \ldots, x_{k+1}^i)^T \); otherwise we stop including a new coefficient into \( X \).

Observing the transform coefficients for different objects, it is concluded that the higher order coefficients can be neglected. Only the first 64x64 elements of the transform matrix are significant for further processing. In order to reduce the dimensionality of the feature vector the above criterion is implemented as follows. The algorithm selects the elements of the first row of the Hadamard transform of an object and calculates the intraclass and interclass distances. Second it takes the elements of the first column of Hadamard transform and computes these distances. In the third step, the algorithm selects the diagonal elements and calculates the distances. It then compares the above results and selects the features which best represent the object of certain class. It was concluded after exhaustive experiments that Hadamard domain coefficients \( \{ H(0,1), (1,1), H(2,2), H(3,3), H(4,4), H(5,5), H(6,6), H(7,7), H(8,8), H(9,9) \} \) are best representative of each class of objects as shown in Figure 2.19a, Figure 2.19b, and Figure 2.19c.
Figure 2.19a Hadamard domain feature vector for an object
<table>
<thead>
<tr>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
<th>21</th>
<th>22</th>
<th>23</th>
<th>24</th>
<th>25</th>
<th>26</th>
<th>27</th>
<th>28</th>
<th>29</th>
<th>30</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>174</td>
<td>174</td>
<td>174</td>
<td>174</td>
<td>240</td>
<td>240</td>
<td>240</td>
<td>240</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
</tr>
<tr>
<td>98</td>
<td>104</td>
<td>104</td>
<td>104</td>
<td>104</td>
<td>106</td>
<td>106</td>
<td>106</td>
<td>106</td>
<td>110</td>
<td>110</td>
<td>110</td>
<td>110</td>
<td>110</td>
<td>110</td>
<td>110</td>
</tr>
<tr>
<td>42</td>
<td>136</td>
<td>136</td>
<td>136</td>
<td>136</td>
<td>202</td>
<td>202</td>
<td>202</td>
<td>202</td>
<td>86</td>
<td>86</td>
<td>86</td>
<td>86</td>
<td>86</td>
<td>86</td>
<td>86</td>
</tr>
<tr>
<td>244</td>
<td>196</td>
<td>196</td>
<td>196</td>
<td>196</td>
<td>244</td>
<td>244</td>
<td>244</td>
<td>244</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
</tr>
<tr>
<td>224</td>
<td>152</td>
<td>152</td>
<td>152</td>
<td>152</td>
<td>196</td>
<td>196</td>
<td>196</td>
<td>196</td>
<td>42</td>
<td>42</td>
<td>42</td>
<td>42</td>
<td>42</td>
<td>42</td>
<td>42</td>
</tr>
<tr>
<td>224</td>
<td>48</td>
<td>148</td>
<td>148</td>
<td>148</td>
<td>176</td>
<td>176</td>
<td>176</td>
<td>176</td>
<td>22</td>
<td>22</td>
<td>22</td>
<td>22</td>
<td>22</td>
<td>22</td>
<td>22</td>
</tr>
<tr>
<td>222</td>
<td>58</td>
<td>58</td>
<td>58</td>
<td>58</td>
<td>96</td>
<td>96</td>
<td>96</td>
<td>96</td>
<td>52</td>
<td>52</td>
<td>52</td>
<td>52</td>
<td>52</td>
<td>52</td>
<td>52</td>
</tr>
<tr>
<td>238</td>
<td>26</td>
<td>26</td>
<td>26</td>
<td>26</td>
<td>40</td>
<td>40</td>
<td>40</td>
<td>40</td>
<td>252</td>
<td>252</td>
<td>252</td>
<td>252</td>
<td>252</td>
<td>252</td>
<td>252</td>
</tr>
<tr>
<td>242</td>
<td>162</td>
<td>162</td>
<td>162</td>
<td>162</td>
<td>96</td>
<td>96</td>
<td>96</td>
<td>96</td>
<td>204</td>
<td>204</td>
<td>204</td>
<td>204</td>
<td>204</td>
<td>204</td>
<td>204</td>
</tr>
<tr>
<td>200</td>
<td>114</td>
<td>114</td>
<td>114</td>
<td>114</td>
<td>52</td>
<td>52</td>
<td>52</td>
<td>52</td>
<td>162</td>
<td>162</td>
<td>162</td>
<td>162</td>
<td>162</td>
<td>162</td>
<td>162</td>
</tr>
<tr>
<td>30</td>
<td>226</td>
<td>226</td>
<td>226</td>
<td>226</td>
<td>34</td>
<td>34</td>
<td>34</td>
<td>34</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>228</td>
<td>22</td>
<td>22</td>
<td>22</td>
<td>22</td>
<td>66</td>
<td>66</td>
<td>66</td>
<td>66</td>
<td>44</td>
<td>44</td>
<td>44</td>
<td>44</td>
<td>44</td>
<td>44</td>
<td>44</td>
</tr>
<tr>
<td>166</td>
<td>104</td>
<td>104</td>
<td>104</td>
<td>104</td>
<td>68</td>
<td>68</td>
<td>68</td>
<td>68</td>
<td>30</td>
<td>30</td>
<td>30</td>
<td>30</td>
<td>30</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>202</td>
<td>110</td>
<td>110</td>
<td>110</td>
<td>110</td>
<td>98</td>
<td>98</td>
<td>98</td>
<td>98</td>
<td>56</td>
<td>56</td>
<td>56</td>
<td>56</td>
<td>56</td>
<td>56</td>
<td>56</td>
</tr>
<tr>
<td>246</td>
<td>106</td>
<td>106</td>
<td>106</td>
<td>106</td>
<td>140</td>
<td>140</td>
<td>140</td>
<td>140</td>
<td>172</td>
<td>172</td>
<td>172</td>
<td>172</td>
<td>172</td>
<td>172</td>
<td>172</td>
</tr>
<tr>
<td>250</td>
<td>102</td>
<td>102</td>
<td>102</td>
<td>102</td>
<td>118</td>
<td>118</td>
<td>118</td>
<td>118</td>
<td>152</td>
<td>152</td>
<td>152</td>
<td>152</td>
<td>152</td>
<td>152</td>
<td>152</td>
</tr>
<tr>
<td>196</td>
<td>132</td>
<td>132</td>
<td>132</td>
<td>132</td>
<td>154</td>
<td>154</td>
<td>154</td>
<td>154</td>
<td>150</td>
<td>150</td>
<td>150</td>
<td>150</td>
<td>150</td>
<td>150</td>
<td>150</td>
</tr>
<tr>
<td>218</td>
<td>158</td>
<td>158</td>
<td>158</td>
<td>158</td>
<td>166</td>
<td>166</td>
<td>166</td>
<td>166</td>
<td>112</td>
<td>112</td>
<td>112</td>
<td>112</td>
<td>112</td>
<td>112</td>
<td>112</td>
</tr>
<tr>
<td>2</td>
<td>130</td>
<td>130</td>
<td>130</td>
<td>130</td>
<td>164</td>
<td>164</td>
<td>164</td>
<td>164</td>
<td>144</td>
<td>144</td>
<td>144</td>
<td>144</td>
<td>144</td>
<td>144</td>
<td>144</td>
</tr>
<tr>
<td>200</td>
<td>176</td>
<td>176</td>
<td>176</td>
<td>176</td>
<td>194</td>
<td>194</td>
<td>194</td>
<td>194</td>
<td>162</td>
<td>162</td>
<td>162</td>
<td>162</td>
<td>162</td>
<td>162</td>
<td>162</td>
</tr>
<tr>
<td>94</td>
<td>196</td>
<td>196</td>
<td>196</td>
<td>196</td>
<td>162</td>
<td>162</td>
<td>162</td>
<td>162</td>
<td>186</td>
<td>186</td>
<td>186</td>
<td>186</td>
<td>186</td>
<td>186</td>
<td>186</td>
</tr>
<tr>
<td>102</td>
<td>202</td>
<td>202</td>
<td>202</td>
<td>202</td>
<td>188</td>
<td>188</td>
<td>188</td>
<td>188</td>
<td>206</td>
<td>206</td>
<td>206</td>
<td>206</td>
<td>206</td>
<td>206</td>
<td>206</td>
</tr>
<tr>
<td>184</td>
<td>218</td>
<td>218</td>
<td>218</td>
<td>218</td>
<td>194</td>
<td>194</td>
<td>194</td>
<td>194</td>
<td>212</td>
<td>212</td>
<td>212</td>
<td>212</td>
<td>212</td>
<td>212</td>
<td>212</td>
</tr>
<tr>
<td>160</td>
<td>214</td>
<td>214</td>
<td>214</td>
<td>214</td>
<td>202</td>
<td>202</td>
<td>202</td>
<td>202</td>
<td>204</td>
<td>204</td>
<td>204</td>
<td>204</td>
<td>204</td>
<td>204</td>
<td>204</td>
</tr>
<tr>
<td>142</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>222</td>
<td>222</td>
<td>222</td>
<td>222</td>
<td>250</td>
<td>250</td>
<td>250</td>
<td>250</td>
<td>250</td>
<td>250</td>
<td>250</td>
</tr>
<tr>
<td>198</td>
<td>242</td>
<td>242</td>
<td>242</td>
<td>242</td>
<td>188</td>
<td>188</td>
<td>188</td>
<td>188</td>
<td>248</td>
<td>248</td>
<td>248</td>
<td>248</td>
<td>248</td>
<td>248</td>
<td>248</td>
</tr>
<tr>
<td>160</td>
<td>60</td>
<td>60</td>
<td>60</td>
<td>60</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>108</td>
<td>108</td>
<td>108</td>
<td>108</td>
<td>108</td>
<td>108</td>
<td>108</td>
</tr>
<tr>
<td>146</td>
<td>84</td>
<td>84</td>
<td>84</td>
<td>84</td>
<td>98</td>
<td>98</td>
<td>98</td>
<td>98</td>
<td>114</td>
<td>114</td>
<td>114</td>
<td>114</td>
<td>114</td>
<td>114</td>
<td>114</td>
</tr>
<tr>
<td>122</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>84</td>
<td>84</td>
<td>84</td>
<td>84</td>
<td>180</td>
<td>180</td>
<td>180</td>
<td>180</td>
<td>180</td>
<td>180</td>
<td>180</td>
</tr>
<tr>
<td>134</td>
<td>134</td>
<td>134</td>
<td>134</td>
<td>134</td>
<td>106</td>
<td>106</td>
<td>106</td>
<td>106</td>
<td>184</td>
<td>184</td>
<td>184</td>
<td>184</td>
<td>184</td>
<td>184</td>
<td>184</td>
</tr>
</tbody>
</table>

**Figure 2.19a Hadamard domain feature vector for an object**
Figure 2.19b Hadamard domain feature vector for an object.
Figure 2.19b Hadamard domain feature vector for an object.
Figure 2.19c Hadamard domain feature vector of an object
<table>
<thead>
<tr>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
<th>21</th>
<th>22</th>
<th>23</th>
<th>24</th>
<th>25</th>
<th>26</th>
<th>27</th>
<th>28</th>
<th>29</th>
<th>30</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>250</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>118</td>
<td>118</td>
<td>118</td>
<td>118</td>
<td>118</td>
<td>118</td>
<td>118</td>
<td>118</td>
<td>118</td>
<td>118</td>
<td>118</td>
<td>118</td>
<td>118</td>
<td>118</td>
<td>118</td>
<td>118</td>
</tr>
<tr>
<td>192</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>194</td>
<td>102</td>
<td>102</td>
<td>102</td>
<td>102</td>
<td>102</td>
<td>102</td>
<td>102</td>
<td>102</td>
<td>102</td>
<td>102</td>
<td>102</td>
<td>102</td>
<td>102</td>
<td>102</td>
<td>102</td>
</tr>
<tr>
<td>110</td>
<td>236</td>
<td>236</td>
<td>236</td>
<td>236</td>
<td>236</td>
<td>236</td>
<td>236</td>
<td>236</td>
<td>236</td>
<td>236</td>
<td>236</td>
<td>236</td>
<td>236</td>
<td>236</td>
<td>236</td>
</tr>
<tr>
<td>110</td>
<td>236</td>
<td>236</td>
<td>236</td>
<td>236</td>
<td>236</td>
<td>236</td>
<td>236</td>
<td>236</td>
<td>236</td>
<td>236</td>
<td>236</td>
<td>236</td>
<td>236</td>
<td>236</td>
<td>236</td>
</tr>
<tr>
<td>86</td>
<td>110</td>
<td>110</td>
<td>110</td>
<td>110</td>
<td>110</td>
<td>110</td>
<td>110</td>
<td>110</td>
<td>110</td>
<td>110</td>
<td>110</td>
<td>110</td>
<td>110</td>
<td>110</td>
<td>110</td>
</tr>
<tr>
<td>152</td>
<td>88</td>
<td>88</td>
<td>88</td>
<td>88</td>
<td>88</td>
<td>88</td>
<td>88</td>
<td>88</td>
<td>88</td>
<td>88</td>
<td>88</td>
<td>88</td>
<td>88</td>
<td>88</td>
<td>88</td>
</tr>
<tr>
<td>150</td>
<td>86</td>
<td>86</td>
<td>86</td>
<td>86</td>
<td>86</td>
<td>86</td>
<td>86</td>
<td>86</td>
<td>86</td>
<td>86</td>
<td>86</td>
<td>86</td>
<td>86</td>
<td>86</td>
<td>86</td>
</tr>
<tr>
<td>44</td>
<td>120</td>
<td>120</td>
<td>120</td>
<td>120</td>
<td>120</td>
<td>120</td>
<td>120</td>
<td>120</td>
<td>120</td>
<td>120</td>
<td>120</td>
<td>120</td>
<td>120</td>
<td>120</td>
<td>120</td>
</tr>
<tr>
<td>42</td>
<td>118</td>
<td>118</td>
<td>118</td>
<td>118</td>
<td>118</td>
<td>118</td>
<td>118</td>
<td>118</td>
<td>118</td>
<td>118</td>
<td>118</td>
<td>118</td>
<td>118</td>
<td>118</td>
<td>118</td>
</tr>
<tr>
<td>40</td>
<td>122</td>
<td>122</td>
<td>122</td>
<td>122</td>
<td>122</td>
<td>122</td>
<td>122</td>
<td>122</td>
<td>122</td>
<td>122</td>
<td>122</td>
<td>122</td>
<td>122</td>
<td>122</td>
<td>122</td>
</tr>
<tr>
<td>162</td>
<td>120</td>
<td>120</td>
<td>120</td>
<td>120</td>
<td>120</td>
<td>120</td>
<td>120</td>
<td>120</td>
<td>120</td>
<td>120</td>
<td>120</td>
<td>120</td>
<td>120</td>
<td>120</td>
<td>120</td>
</tr>
<tr>
<td>160</td>
<td>118</td>
<td>118</td>
<td>118</td>
<td>118</td>
<td>118</td>
<td>118</td>
<td>118</td>
<td>118</td>
<td>118</td>
<td>118</td>
<td>118</td>
<td>118</td>
<td>118</td>
<td>118</td>
<td>118</td>
</tr>
<tr>
<td>132</td>
<td>186</td>
<td>186</td>
<td>186</td>
<td>186</td>
<td>186</td>
<td>186</td>
<td>186</td>
<td>186</td>
<td>186</td>
<td>186</td>
<td>186</td>
<td>186</td>
<td>186</td>
<td>186</td>
<td>186</td>
</tr>
<tr>
<td>130</td>
<td>188</td>
<td>188</td>
<td>188</td>
<td>188</td>
<td>188</td>
<td>188</td>
<td>188</td>
<td>188</td>
<td>188</td>
<td>188</td>
<td>188</td>
<td>188</td>
<td>188</td>
<td>188</td>
<td>188</td>
</tr>
<tr>
<td>198</td>
<td>194</td>
<td>194</td>
<td>194</td>
<td>194</td>
<td>194</td>
<td>194</td>
<td>194</td>
<td>194</td>
<td>194</td>
<td>194</td>
<td>194</td>
<td>194</td>
<td>194</td>
<td>194</td>
<td>194</td>
</tr>
<tr>
<td>200</td>
<td>192</td>
<td>192</td>
<td>192</td>
<td>192</td>
<td>192</td>
<td>192</td>
<td>192</td>
<td>192</td>
<td>192</td>
<td>192</td>
<td>192</td>
<td>192</td>
<td>192</td>
<td>192</td>
<td>192</td>
</tr>
<tr>
<td>172</td>
<td>170</td>
<td>170</td>
<td>170</td>
<td>170</td>
<td>170</td>
<td>170</td>
<td>170</td>
<td>170</td>
<td>170</td>
<td>170</td>
<td>170</td>
<td>170</td>
<td>170</td>
<td>170</td>
<td>170</td>
</tr>
<tr>
<td>230</td>
<td>176</td>
<td>176</td>
<td>176</td>
<td>176</td>
<td>176</td>
<td>176</td>
<td>176</td>
<td>176</td>
<td>176</td>
<td>176</td>
<td>176</td>
<td>176</td>
<td>176</td>
<td>176</td>
<td>176</td>
</tr>
<tr>
<td>34</td>
<td>140</td>
<td>140</td>
<td>140</td>
<td>140</td>
<td>140</td>
<td>140</td>
<td>140</td>
<td>140</td>
<td>140</td>
<td>140</td>
<td>140</td>
<td>140</td>
<td>140</td>
<td>140</td>
<td>140</td>
</tr>
<tr>
<td>36</td>
<td>142</td>
<td>142</td>
<td>142</td>
<td>142</td>
<td>142</td>
<td>142</td>
<td>142</td>
<td>142</td>
<td>142</td>
<td>142</td>
<td>142</td>
<td>142</td>
<td>142</td>
<td>142</td>
<td>142</td>
</tr>
<tr>
<td>198</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>200</td>
</tr>
<tr>
<td>198</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>200</td>
</tr>
</tbody>
</table>

Figure 2.19c Hadamard domain feature vector of an object
In order to train the classifier, individual objects are exposed to the camera and 2-D FHT is performed in the image area that contains the object of interest. This gives a feature vector for each object which is stored in the database. Following this training phase, a cluster of objects is presented to the system. The system isolates each object by using a parallel region growing and labeling algorithm and extracts features for each unknown object. It then classifies the objects using the minimum-distance rule. The classifier calculates the Euclidean distance between the unknown pattern vector $X_0$ (i.e., a multidimensional vector having the selected matrix elements as its coordinates) and center $Z_i$ of pattern classes is given as

$$|d_{0,i}| = |X_0 - Z_i^i| = \sum_{j=1}^{n} \sqrt{(x_{0j}^j - z_{ij})^2}, \quad \forall i$$

where $n$ is the dimension of the feature vector. The pattern $X_0$ is classified belonging to pattern class $m$ provided $|d_{0,m}|$ is minimum.

After performing the above steps for object recognition the results obtained (as shown in Chapter 4) dictate a high noise immunity of the proposed classifier.
CHAPTER 3

DEVELOPMENT AND IMPLEMENTATION OF 2-D FHT ON PARALLEL PROCESSOR

The algorithms required for object recognition in a computer vision system are computationally intensive. They require a large amount of data to be processed in a real-time application. This level of processing power is only possible with a high degree of parallelism, which has not been available cost-effectively in the past. With the availability of several commercial parallel processors, massive parallelism has been applied to different pattern recognition problems.

Of the various parallel processing models available, the distributed memory model is the most commonly used architecture. This model assumes that each processor has its own memory and communicates with others by the dedicated channels. Hypercube is one form of such architecture which is constructed from $N = 2^n$ identical processors connected through point-to-point bidirectional links. The order of an $N$ processor hypercube is given by $n$ which is referred to as an $n$-cube. It takes at most $n$ steps for one processor to communicate with other processors or nodes. Usually a hypercube array communicates with the outside world through a
A hypercube of dimension \( n \) is constructed by assigning a unique binary number to each node processor according to the following recursive rule of the gray-code. First, we prefix the node labels in one of the \((n-1)\) cubes with a 0 such that they are of the form \(0xx...xx\). Second, prefix the node labels in the other \((n-1)\) cube with a 1 such that they are of the form \(1xx...xx\). Finally, we connect the two \((n-1)\) cubes with communication links between nodes that have labels differing only in their most significant bit. Figure 3.1a shows a 16-node hypercube system and Figure 3.1b shows its mapping for a ring topology.

Among several networks, mesh (nearest-neighbor) and ring are two well-known topologies that can be mapped onto a hypercube system. The selection of a particular topology depends upon the problem in question. In this research, the Hadamard transform is implemented on a 16-node ring topology mapped onto a 16-node hypercube machine. The selection of ring topology is explained in the subsequent section.

To implement the 2-D FHT on a parallel processor, we use the separability property of the Hadamard transform. This property allows to obtain the 2-D FHT in column major or row major order. Since the input image consists of \(N\times N\) data...
Figure 3.1a 16-node hypercube

Figure 3.1b 16-node ring mapped onto 16-node hypercube
points, it requires \( N \) processing elements each capable of performing one \( N \)-point fast Hadamard transform. A 1-D FHT is performed on each of the \( N \) rows producing an \( N \)-element output data sequence. The above transform results in \( NxN \) intermediate data matrix. A 1-D FHT algorithm is again applied to each of the \( N \) columns to obtain the 2-D FHT of the original input image as shown in Figure 3.2. The same results would have been obtained by first taking the transform along the columns of the input data and then along the rows of the result.

3.1 Topology Selection

Each node of the Ametek S-14 includes an 80286 microprocessor as its data processing unit. In addition, each node has a separate 80186 processor to handle internode message passing and 1M bytes of memory. Up to 256 nodes can be incorporated in the system at a peak performance of 15 MegaFlops. The processing elements of the Ametek S-14 can be configured in mesh or ring topology.

In order to download the input data from the parent processor (VAX 11/750) into 16 child processors in mesh topology, input data has to be rearranged according to the gray code pattern of the mesh topology. Since the input data is a 512x512 matrix, it would take a considerable amount of time to rearrange it into a specific pattern. To save this time, ring topology is used in this research. The ring
Figure 3.2 Computation of 2-D Hadamard transform as a series of 1-D transforms
topology does not require any reordering of input data because there is a one to one mapping of the input data to respective nodes.

A single download instruction of the S-14 concurrent processor downloads the preprocessed binary image from the host to each of the nodes of the parallel processor. This is done by first partitioning the preprocessed binary image of size 512x512 into 16 equal parts or section in a row-major form resulting in 16 two-dimensional arrays of size 32x512. This is shown in Figure 3.3. Each subimage is then mapped onto a separate node in the concurrent processor by using the download() instruction of the S-14. Each node of the Ametek S-14 receives a one-dimensional array of size 16,384 elements.

3.2 Memory Management of S-14

The Ametek S-14 has constraints on memory in each node. These constraints are imposed by the Crystalline operating system (XOS), which controls host to node and the node to node communications. The operating system allocates 64k bytes for the static memory, 32k bytes for the stack in each node, and 30k bytes for its own internal use. The remaining 850k bytes are available as dynamic memory. This memory can be used during run time. Two instructions provided for allocating dynamic memory are mmalloc() for array lengths less than 64k bytes and bmalloc() for array lengths greater than 64k bytes.
Figure 3.3 Image partitioning and mapping on S-14
To obtain a high precision computation of the 2-D FHT, the results are stored as double precision numbers, each of which occupies 8 bytes of memory. The algorithm uses two arrays each having 16,384 double precision numbers which takes 2x16384x8 bytes or 256k bytes of dynamic memory. Two buffers of 64k bytes each are used to hold the intermediate results during broadcast. The mark() and mfree() instructions are used to allocate and deallocate a contiguous block of dynamic memory. These commands are always used in pairs with the first mfree() freeing the last marked memory block.

3.3 FHT Implementation on S-14

A general flow diagram of the 2-D FHT implementation on the S-14 is shown in Figure 3.4. The host processor downloads the preprocessed binary image into 16 nodes of S-14, with each child receiving an array of size 32x512. Each node performs 32 1-D FHT's simultaneously on a 512 point array of input data resulting in an intermediate array. To rearrange the data among each of the 16 nodes, we use the broadcast instruction of the S-14 system. The syntax of the broadcast statement is

```
brdcstal(real, ntermsn, sender);
```

This instruction broadcasts an array of long integers, characters or double precision numbers having ntermsn elements from sender node to every other node in the S-14. The maximum
Download subimages of size 32x512 to the corresponding nodes of concurrent processor

Perform 32 1-D FHT each on 512 elements in every node

Broadcast 32x512 subimages from each node to every other node

Rearrange (i.e. transpose) results of 1-D transform in each node

Perform 32 1-D FHT of length 512 elements in each node to obtain the Hadamard domain coefficients.

Upload the Hadamard transformed 512x512 image to the parent processor

Stop

Figure 3.4 A general flow-diagram of the 2-D FHT on System S-14
number of bytes that can be broadcasted using this instruction should be less than 63k-1. This means that to broadcast an array of 16,384 double precision numbers (128k bytes) three such broadcasts are needed. This problem is circumvented by converting the double precision numbers to long integers which reduces the number of broadcast statements to two per node. To broadcast an array of 16,384 long integers we use the following routine

```c
user_main()
{
    int ntermsn, sender;
    long *buff;
    ntermsn = 8192;
    sender = childnum;

    brdcstal(buff, ntermsn, sender);
    brdcstal(&buff[8192], ntermsn, sender);
}
```

Finally 32 1-D FHT transforms are performed on the arranged data in each node to obtain the 2-D Hadamard transform of the object of interest in the given image. These transform domain coefficients are uploaded to the host system using the upload instruction of the Ametek S-14. Once the
Hadamard domain coefficients are obtained, the object feature vector is extracted using the adaptive algorithm described in Chapter 2. Using the minimum-distance criterion, the unknown objects are classified in accordance with stored prototypes in the database. The results of classification are given in Chapter 4. The speedup of the parallel implementation is provided in the next section.

### 3.4 Fast Hadamard Transform Timing Analysis

The speedup factor \( S \) can be defined as the ratio of the time taken for a particular algorithm or procedure to be executed on a single processor to the time taken to perform the same task on a parallel machine. It is given by

\[
S = \frac{t_{\text{seq}}}{t_{\text{parallel}}}
\]  

(3.1)

where \( t_{\text{seq}} \) is the time for the sequential implementation in one node of the concurrent system and \( t_{\text{parallel}} \) is the time for the parallel processor implementation, both being measured with respect to the sequential processor internal clock.

The time for sequential implementation of the 2-D FHT is given by

\[
t_{\text{seq}} = t_{1-D,\text{FHT}} + t_{\text{transpose}} + t_{1-D,\text{FHT}}
\]  

(3.2)

where \( t_{1-D,\text{FHT}} \) is the time taken to perform 1-D FHT and \( t_{\text{transpose}} \).
is the time taken to transpose the row-transformed image. Since the time \( t_{\text{transpose}} \) is negligible as compared to \( t_{1-D_{\text{FHT}}} \), the time for sequential implementation is

\[
t_{\text{seq}} = 2x t_{1-D_{\text{FHT}}}
\]

Similarly, the parallel implementation of the 2-D FHT is given by

\[
t_{\text{parallel}} = t_{\text{dnld}} + t_{1-D_{\text{FHT}}} + t_{\text{transpose}} + t_{1-D_{\text{FHT}}} + t_{\text{upld}} \quad (3.3)
\]

where

- \( t_{1-D_{\text{FHT}}} \) is the time taken to perform 1-D FHT in one node of the hypercube system,
- \( t_{\text{dnld}} \) is the time taken to download 32x512 subimages from parent to corresponding nodes of the parallel processor,
- \( t_{\text{transpose}} \) is the time taken to transpose the image using inter-node communications,
- \( t_{\text{upld}} \) is the time taken to upload the resultant Hadamard transform coefficients from nodes to the parent processor.

The upload \((t_{\text{dnld}})\) and download \((t_{\text{upld}})\) time play a significant role in the calculation of speedup factor. This is due to the low I/O bandwidth between the VAX and System S-14, which allows only a maximum of 64k bytes of data transfer in a single communication. The node-to-node communication time for S-14 is \( t_b = 10 \) microseconds (which is equivalent to transferring one byte of data between two nodes). The overhead time \( t_{\text{over}} \) which is a characteristic of the machine and the operating system for S-14 is equal to 300 microseconds. As
seen from the timing results presented in the next Chapter, the communication time far exceeds the actual processing time if the communication between node and host are taken into account. A better speedup could be achieved by neglecting the data upload and download time.
The Hadamard transform is implemented on the Ametek S-14 using the VAX 11/750 minicomputer running under the UNIX 4.3 BSD operating system as the host processor. Image acquisition, digitization, and display are performed through the VICOM image processor, which is interfaced with an RCA camera and controlled by the VAX computer. The overall experimental environment is shown in Figure 4.1.

Parent and child programs were written for ring topology. These programs were developed using the user topology interface routines of S-14. All aspects of the processor coordination and communications were programmed. Each of the parallel programs was coded in the C programming language with the aid of the ADE (Ametek Development Environment) software tools.

4.1 Results of Histogramming and Thresholding

In Figure 4.2 a gray level image of an object is shown and Figure 4.3 gives its gray-level histogram. This distribution is used to select a threshold value of 45 giving
Figure 4.1 Experimental set-up
Figure 4.2 Gray-level image of an object
Figure 4.3 Gray-level histogram of an object

Figure 4.4 Binary image of an object
a noise free binary image as shown in Figure 4.4.

4.2 Results of Object Translation and Rotation

A coordinate axis transformation is applied to the binary image of Figure 4.4 to obtain a translated (Figure 4.5) and rotated (Figure 4.6) image. The results of both translation and rotation of the given object are shown in Figure 4.7. From this it could be concluded that the proposed algorithm can be satisfactorily used for object translation and rotation.

4.3 Results of Parallel Implementation of FHT

Parallel implementation of the fast Hadamard transform resulted in a significant speedup as compared to the sequential algorithm. Timing results for the serial and parallel implementation of the fast Hadamard transform are given in Table 4.1. The timing results for parallel implementation are obtained by excluding the host-to-node and I/O communications.

4.4 Results of Object Recognition and Classification

Once the Hadamard domain coefficients are obtained for each object used in the experiments, a cluster of objects is presented to the system (see Figure 4.8). Each object is isolated by using a parallel region growing and labeling algorithm. The classifier then identifies each object using a
Figure 4.5 Translated image of an object

Figure 4.6 Rotated image of an object
Figure 4.7 Gray-level image of a cluster of objects
All the times are in seconds.

<table>
<thead>
<tr>
<th>t_{seq}</th>
<th>t_{old}</th>
<th>t_{upld}</th>
<th>t_{in-FHT}</th>
<th>t_{trans}</th>
<th>t_{total}</th>
<th>(S)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1980.3</td>
<td>14.1</td>
<td>14.4</td>
<td>28.6</td>
<td>140.5</td>
<td>197.7</td>
<td>10.01</td>
</tr>
</tbody>
</table>

Table 4.1 Speedup analysis of parallel implementation of 2-D FHT

<table>
<thead>
<tr>
<th></th>
<th>Nut</th>
<th>Key</th>
<th>Plier</th>
<th>Bolt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nut</td>
<td>3.1</td>
<td>169.5</td>
<td>232.1</td>
<td>386.9</td>
</tr>
<tr>
<td>Key</td>
<td>87.3</td>
<td>5.2</td>
<td>151.4</td>
<td>268.2</td>
</tr>
<tr>
<td>Plier</td>
<td>51.9</td>
<td>84.3</td>
<td>6.4</td>
<td>178.1</td>
</tr>
<tr>
<td>Bolt</td>
<td>136.9</td>
<td>202.5</td>
<td>278.1</td>
<td>9.4</td>
</tr>
</tbody>
</table>

Table 4.2 Classification results
minimum-distance rule. The results of this classification are listed in Table 4.2. From the experimental results obtained, it is concluded that the Hadamard domain features are sufficient for recognizing different objects.
CHAPTER 5
CONCLUSION AND DISCUSSION

In this research, a parallel object recognition algorithm using Hadamard domain features has been implemented on a hypercube system. The proposed method extracts the first 8 to 10 Hadamard domain coefficients as the characteristic features of a given object in such a way that they are independent of objects position, orientation, and size. Gray-level invariance is achieved by using binary images of the objects. For the set of objects used in experiments the system gave 99.8% accurate classification results.

A tenfold speedup is achieved by parallelizing the fast Hadamard transform for feature extraction. This speedup is obtained by decomposing the given image into 16 subimages and distributing them among 16 processors of the hypercube machine mapped in ring topology.

Although the feature vector obtained using the Hadamard transform resulted in relatively good classification, the advantages and limitations of the Hadamard transform for other pattern-recognition tasks remain to be explored. At present it appears that it will be more useful in the high-speed processing of simple patterns (i.e. no depth information, plane surfaces, and
regular illumination etc.) then in the analysis of more complex patterns.

Fast computation of Hadamard domain features using a parallel architecture allow their use in realtime applications such as manufacture visual inspection. The proposed scheme can be extended to the inspection of 3-D objects in an industrial environment. This could be achieved by obtaining three 2-dimensional images of the object viewed in three orthogonal directions.
REFERENCES


8. J. Carl and R. Swartwood, "A hybrid Walsh transform


16. Z. Fang, X. Li, and L. Ni, "Parallel algorithms for


APPENDIX A

The purpose of this appendix is to explain the use of VICOM image processor and how to classify an unknown object using the algorithms developed in this research.

The notation used is that capital letters after the VICOM console prompt (*) and VAX console prompt (>) are the actual commands typed and small letters gives an explanation for that instruction. The following steps are taken to obtain an ascii image.

First the given object is presented to an RCA 2000 camera. There are three frame buffers namely CAM1, CAM2, and CAM3. Any of these buffers can be used to capture a particular view of the given object. Type the following commands from the VICOM console i.e.,

* > CAM [1, or 2, or 3] Three frame buffers.

* > DIG [1, or 2, or 3] Digitize the given image.

In order to store these images in the home directory type "vsh" command from VAX console. After doing that press Shift, Esc, and 0 keys simultaneously from the VICOM console. This connects VAX and the image processor. From the VAX console and
in the home directory type the following command
> WIM [1, or 2, or 3] object1.img
Name of the file to store the ascii image. (The filename could be any combination of letters but the file extension should always be .img)
> RIM [1, or 2, or 3] object1.img
This command displays the processed image file onto the VICOM video screen.

Once we obtain an ascii image, the next step is preprocess that image using the following algorithm,
> FTR.C
Algorithm for smoothing, histogramming, translation and rotation of the given image.

If the scene consists of more than one object we used the following algorithm to segment the given image into each object having a unique label.
> REGIONP.C AND REGIONC.C
Parallel region growing and labelling algorithm.

The next step is to apply the Hadamard transform algorithms developed on the Ametek S-14 parallel processor i.e.,
> FHTP.C AND FHTC.C
Parallel algorithms to obtain Hadamard domain coefficients for
each preprocessed image.

In order to normalize and further reduce the dimensionality of the feature vector we used, > DIST1.C Normalizes the Hadamard domain coefficients, and obtains a feature vector by an adaptive algorithm.

Once the feature vector for each object is obtained next step is to classify them accordingly using the minimum distance rule i.e, > CLASSIFY.C Algorithm to classify each unknown object depending upon its feature vector.
Appendix B

/* THIS IS A PROGRAM CONSITING OF */
/* */
1: ALGORITHMS FOR SMOOTHING AND HISTOGRAMMING
/* */
/* */
2: ALGORITHMS FOR FINDING THE AREA, CENTROID,
/* */
FIRST AND SECOND MOMENTS, TRANSLATION AND
/* */
ROTATION OF AN OBJECT
/* */
/* */
SOFTWARE DEVELOPED BY:
/* */
SAIFUDDIN MOIZ
/* */

#include <stdio.h>
#include <math.h>

char image[512][512],new[512][512],b[512][512];
double sqrt();
FILE *fpout;
void Collec();
void Smooth();
void Hist();
void Trans();
void Rotate();

main()
{
    int i,j,rows,cols;
    char infile[25];

    Collec(&rows,&cols,infile);
    Smooth(image);
    Hist(image);
    Trans(image);
    Rotate(image);
}

/* Function for collecting the input information */
/* */
/* */
/* */
void Collec(x,y,z)
int *x,*y;
char z[25];
{
    int i,j;
    FILE *fpinp;

    printf("\n ENTER INPUT FILE NAME :");
    scanf("%s",z);
    printf("\n ENTER FILE SIZE :\n");
    scanf("%d",x);
    printf("\n NO. OF ROWS :");
    scanf("%d",y);
    printf("\n NO. OF COLUMNS :");
    scanf("%d",y);
    printf("\nROWS :%d COLS :%d FILE :%s
",*x,*y,z);
    fpinp = fopen(z,"r");
    fread(image,512*512,1,fpinp);
    printf("\nDone with reading the image
");
    fclose(fpinp);
}

void Hist(hist)
    char hist[512][512];
{
    int i,j,k,th,h[256],max;
    FILE *fpout;
    char name[25];

    printf("\n ENTER BINARY FILE NAME :");
    scanf("%s",name);
    printf("%s
",name);

    printf("\n HISTOGRAMMING AND THRESHOLDING \n");
    max = 0;
    for(i=0;i<=511;i++)
    {
        for(j=0;j<=511;j++)
        {
            if(hist[i][j]>max)
                max = hist[i][j];
        }
    }
    printf("\n MAX VALUE = %d\n",max);
    for(i=0;i<=255;i++)

h[i] = 0;
for(i=0;i<=511;i++)
{
    for(j=0;j<=511;j++)
    {
        k = hist[i][j];
        h[k]++;
    }
}
for(i=0;i<=max;i++)
printf("h[%6d]=%6d :\n",i,h[i]);
printf("\n ENTER THE DESIRED THRESHOLD VALUE =");
scanf("%d",&th);
for(i=0;i<=511;i++)
{
    for(j=0;j<=511;j++)
    {
        if(hist[i][j]>=th)
            hist[i][j] = 60;
        else    hist[i][j] = 0;
    }
}
for(i=465;i<=511;i++)
{
    for(j=0;j<=511;j++)
    {
        hist[i][j] = 60;
    }
}

fpout = fopen(name,"w");
fwrite(hist,512*512,1,fpout);
fclose(fpout);

/**************************************************************************/
/* FUNCTION FOR SMOOTHING THE IMAGE */
/**************************************************************************/
void Smooth(b)
char b[][512];
{
    int i,j;
    int b1,b2,b3;
    printf("\n SMOOTHING \n");

    for(i=1;i<=510;i++)
    {
        for(j=1;j<=510;j++)
```c
{ 
  b1 = b[i-1][j-1]+b[i-1][j]+b[i-1][j+1];
  b2 = b[i][j-1]+b[i][j]+b[i][j+1];
  b3 = b[i+1][j-1]+b[i+1][j]+b[i+1][j+1];
  b[i][j] = (b1+b2+b3)/9.0;
}

/*******************************************************************************/
/*                                                                            */
/* FUNCTION FOR TRANSLATING THE OBJECT TO THE                                  */
/* CENTER OF THE IMAGE FRAME                                                  */
/*                                                                            */
/*******************************************************************************/
void Trans (trans) 
{
  /* assignment of variables */
  char name1[25];
  double pi = 4*atan(1.0);
  int centi, centj, i, j, v, newi, newj, count, diffi, diffj;
  long area = 0, I = 0, J = 0, Isqr = 0, Jsqr = 0, IJ = 0;
  double theta, value, a, b, c, slope, theta1, ftheta, r;
  FILE *fpout;

  printf("Enter TRANSLATED FILE NAME : ");
  scanf("%s", name1);
  printf("%s
", name1);

  i = 511;
  while(i != 0)
  {
    for(j=0; j<=511; j++)
    {
      if(trans[i][j] == 0)
      {
        v = 511-i;
        area++;
        I+= v;
        J+= j;
      }
    }
    i = i-1;
  }

  printf("the area is equal to %d square units
", area);
```
centi = I/area;
centj = J/area;

printf("the coordinates of centroid are (%3d,%3d) \n", centi, centj);
printf("\n");
printf("The translation is being done here...\n");

for(i=0;i<=511;i++)
{
    for(j=0;j<=511;j++)
    {
        new[i][j] = 60;
    }
}

if(centi<=255 && centj<=255)
{
    diffi = 255-centi;
    diffj = 255-centj;
    for(i=0;i<=511;i++)
    {
        for(j=0;j<=511;j++)
        {
            if(trans[i][j] == 0)
            {
                newi = i+diffi;
                newj = j+diffj;
                new[newi][newj] = 0;
            }
        }
    }
}
else if(centi<=255 && centj>255)
{
    diffi = 255-centi;
    diffj = centj-255;
    for(i=0;i<=511;i++)
    {
        for(j=0;j<=511;j++)
        {
            if(trans[i][j] == 0)
            {
                newi = abs(i-diffi);
                newj = abs(j-diffj);
                new[newi][newj] = 0;
            }
        }
    }
}
else if(centi>255 && centj<=255)
{
    diffi = centi-255;
printf("the value of diffi = %d\n", diffi);
diffj = 255-centj;
printf("the value of diffj = %d\n", diffj);
for(i=0; i<=511; i++)
{
    for(j=0; j<=511; j++)
    {
        if(trans[i][j] == 0)
        {
            newi = abs(i-diffi);
            newj = j+diffj;
            new[newi][newj] = 0;
        }
    }
}
else if(centj>255 & centj>255)
{
    diffi = centi-255;
    diffj = centj-255;
    for(i=0; i<=511; i++)
    {
        for(j=0; j<=511; j++)
        {
            if(trans[i][j] == 0)
            {
                newi = abs(i-diffi);
                newj = abs(j-diffj);
                new[newi][newj] = 0;
            }
        }
    }
}
printf("the value of new (255,255) = %d\n", new[255][255]);
fpout = fopen(name1,"w");
fwrite(new,512*512,1,fpout);
fclose(fpout);
I = 0; J = 0; area = 0; v = 0;
printf("The area and centroid of translated image are\n");
i = 511;
while(i != 0)
{
    for(j=0; j<=511; j++)
    {
        if(new[i][j] == 0)
        {
            v = 511-i;
            area++; i+=v; J+= j;
printf("the area is equal to %d square units\n",area);

centi = I/area;
centj = J/area;

printf("the coordinates of centroid are (%3d,%3d)\n",centi,centj);
printf("\n");
/*for(i=450;i<=511;i++)
{
    for(j=0;j<=511;j++)
    {
        printf("new= [%d]\n", new[i][j]);
    }
}
*/

*/

*******************************************************************************/
/*
* FUNCTION THAT ALIGNS THE PRINCIPAL AXIS OF ANY GIVEN
* OBJECT IN PARALLEL WITH THE Y-COORDINATE AXIS OF THE
* FRAME OF REFERENCE. THE ROTATION IS ABOUT THE
* CENTROID OF THE GIVEN OBJECT.
*/
*******************************************************************************/

void Rotate(rotate)
char rotate[512][512];
{
    /* assignment of variables */
double pi = 4*atan(1.0);
int centi, centj, i, j, v,newi,newj,count;
long area = 0, I = 0, J = 0, Isqr = 0, Jsqr = 0, IJ = 0;
double theta, value, a, b, c,slope,thetal,ftheta,r;
char name2[25];
FILE *fpout;

printf("\n ENTER ROTATED FILE NAME :");
scanf("%s",name2);
printf("\n",name2);

for(i=0;i<=511;i++)
{
    for(j=0;j<=511;j++)
    {
        new[i][j] = 60;
if (rotate[i][j] == 0)
{
    v = 511-i;
    area++;
    I+= v;
    J+= j;
}
}
i=i-1;

printf("the area is equal to %d square units\n",area);

centi = I/area;
centj = J/area;

printf("the coordinates of centroid are (%3d,%3d)\n",centi,centj);
printf("\n");
i = 511;
while(i != 0)
{
    for(j=0;j<=511;j++)
    {
        if(rotate[i][j] == 0)
        {
            v = 511-i;
            Isqr+= (v-centi)*(v-centi);
            Jsqr+= (j-centj)*(j-centj);
            IJ+= (v-centi)*(j-centj);
        }
    }
    i=i-1;
}

a = Isqr;
b = 2*IJ;
c = Jsqr;

value = b/(a-c);
theta = atan(value);
printf("the first moment along x axis is equal to %d \n",I);
printf("the first moment along y axis is equal to %d \n",J);
printf("the second moment along x axis is equal to %d \n",Isqr);
printf("the second moment along y axis is equal to %d \n",Jsqr);
printf("the second moment along the diagonal axis is equal to %d \n",IJ);
printf("the value is equal to %f \n",value);
printf("the value of tan twotheta is equal to %f radians\n",theta);
value = (180/pi)*0.5*theta;
printf("the value of theta is equal to %f degrees\n",value);
printf("\n");

/* the new centroid values are given here */

centi = 511-centi;
centj = centj;
printf("%d  %d\n",centi,centj);

/* the value of the angle of rotation is calculated below */
theta = (pi/2) - (theta/2);
printf("theta = %f\n",theta);

/* the comparision of the diagonal moment ie. the 1,1 moment is */
/* done here and the corresponding rotation implemented */

if(IJ > 0)
{
    /* the exact rotation is obtained here */
    v = 0;
}
for(i=0;i<=511;i++)
{
    for(j=0;j<=511;j++)
    {
        if(rotate[i][j] == 0)
        {
            if(i<centi && j>centj)
            {
                a = centi-i;
                b = centj-j;
                slope = a/b;

                thetal = fabs(atan(slope));
                ftheta = theta + thetal;

                \( r = (\text{centi}-i)(\text{centi}-i) + (\text{centj}-j)(\text{centj}-j); \)
                r = sqrt(r);

                newj = centj + r*cos(ftheta) + 0.5;
                newi = centi - r*sin(ftheta) + 0.5;

                new[newi][newj] = 0;
            }

            else if(i<centi && j<centj)
            {
                a = centi-i;
                b = centj-j;
                slope = a/b;

                thetal = fabs(atan(slope));
                ftheta = thetal - theta;

                \( r = (\text{centi}-i)(\text{centi}-i) + (\text{centj}-j)(\text{centj}-j); \)
                r = sqrt(r);

                newj = centj - r*cos(ftheta) + 0.5;
                newi = centi - r*sin(ftheta) + 0.5;

                new[newi][newj] = 0;
            }

            else if(i>centi && j<centj)
            {
                a = centi-i;
                b = centj-j;
                slope = a/b;

                thetal = fabs(atan(slope));
ftheta = theta + thetal;
(r = (centi-i)*(centi-i) +
(centj-j)*(centj-j));

r = sqrt(r);

newj = centj - r*cos(ftheta) + 0.5;
newi = centi + r*sin(ftheta) + 0.5;

new[newi][newj] = 0;
}

else if(i>centi && j>centj)
{
    a = centi-i;
b = centj-j;
slope = a/b;

    thetal = fabs(atan(slope));
ftheta = thetal - theta;

(centj-j)*(centj-j);

r = sqrt(r);

newj = centj + r*cos(ftheta) + 0.5;
newi = centi + r*sin(ftheta) + 0.5;

new[newi][newj] = 0;
}

else if(i==centi && j>centj)
{
    ftheta = theta;

(centj-j)*(centj-j);

r = sqrt(r);

newj = centj + r*cos(ftheta) + 0.5;
newi = centi - r*sin(ftheta) + 0.5;

new[newi][newj] = 0;
}

else if(i<centi && j==centj)
{
    ftheta = theta;

(centj-j)*(centj-j);

r = sqrt(r);

}
newj = centj - r*cos(ftheta) + 0.5;
newi = centi - r*sin(ftheta) + 0.5;

new[newi][newj] = 0;
}

else if(i==centi && j<centj)
{
    ftheta = theta;
    r = (centi-i)*(centi-i) + (centj-j)*(centj-j);
    r = sqrt(r);
    newj = centj - r*cos(ftheta) + 0.5;
    newi = centi + r*sin(ftheta) + 0.5;
    new[newi][newj] = 0;
}

else if(i>centi && j==centj)
{
    ftheta = theta;
    r = (centi-i)*(centi-i) + (centj-j)*(centj-j);
    r = sqrt(r);
    newj = centj + r*cos(ftheta) + 0.5;
    newi = centi + r*sin(ftheta) + 0.5;
    new[newi][newj] = 0;
}

else if(i==centi && j==centj)
{
    new[centi][centj] = 0;
}

/* the exact rotation is obtained here */
v = 0;
for(i=0;i<=511;i++)
{
    for(j=0;j<=511;j++)
    {
        if(rotate[i][j] == 0)
        {
            if(i<centi && j>centj)
            {
                a = centi-i;
                b = centj-j;
                slope = a/b;

                thetal = fabs(atan(slope));
                ftheta = thetal - theta;

                r = (centi-i)*(centi-i) + (centj-j)*(centj-j);
                r = sqrt(r);

                newj = centj + r*cos(ftheta) + 0.5;
                newi = centi - r*sin(ftheta) + 0.5;

                new[newi][newj] = 0;
            }

            else if(i<centi && j<centj)
            {
                a = centi-i;
                b = centj-j;
                slope = a/b;

                thetal = fabs(atan(slope));
                ftheta = thetal + theta;

                r = (centi-i)*(centi-i) + (centj-j)*(centj-j);
                r = sqrt(r);

                newj = centj - r*cos(ftheta) + 0.5;
                newi = centi - r*sin(ftheta) + 0.5;

                new[newi][newj] = 0;
            }

            else if(i>centi && j<centj)
            {
                a = centi-i;
                b = centj-j;
                slope = a/b;


\[ \theta_1 = \text{fabs}(\text{atan}(\text{slope})); \]
\[ \phi_1 = \theta_1 - \theta; \]
\[ r = (\text{centi-i})*(\text{centi-i}) + (\text{centj-j})*(\text{centj-j}); \]
\[ r = \sqrt{r}; \]
\[ \text{newj} = \text{centj} - r*\cos(\phi_1) + 0.5; \]
\[ \text{newi} = \text{centi} + r*\sin(\phi_1) + 0.5; \]
\[ \text{new[newi][newj]} = 0; \]

else if(i>\text{centi} \&\& j>\text{centj})
{
 a = \text{centi-i};
 b = \text{centj-j};
\text{slope} = a/b;

\[ \theta_1 = \text{fabs}(\text{atan}(\text{slope})); \]
\[ \phi_1 = \theta_1 + \theta; \]
\[ r = (\text{centi-i})*(\text{centi-i}) + (\text{centj-j})*(\text{centj-j}); \]
\[ r = \sqrt{r}; \]
\[ \text{newj} = \text{centj} + r*\cos(\phi_1) + 0.5; \]
\[ \text{newi} = \text{centi} + r*\sin(\phi_1) + 0.5; \]
\[ \text{new[newi][newj]} = 0; \]
}

else if(i==\text{centi} \&\& j>\text{centj})
{
 f\theta_1 = \theta; \]
\[ r = (\text{centi-i})*(\text{centi-i}) + (\text{centj-j})*(\text{centj-j}); \]
\[ r = \sqrt{r}; \]
\[ \text{newj} = \text{centj} + r*\cos(\phi_1) + 0.5; \]
\[ \text{newi} = \text{centi} + r*\sin(\phi_1) + 0.5; \]
\[ \text{new[newi][newj]} = 0; \]
}

else if(i<\text{centi} \&\& j==\text{centj})
{
 f\theta_1 = \theta; \]
\[ r = (\text{centi-i})*(\text{centi-i}) + (\text{centj-j})*(\text{centj-j}); \]
\[ r = \sqrt{r}; \]
\[ \text{newj} = \text{centj} + r*\cos(\phi_1) + 0.5; \]
\[ \text{newi} = \text{centi} + r*\sin(\phi_1) + 0.5; \]
\[ \text{new[newi][newj]} = 0; \]
}

else if(i<\text{centi} \&\& j==\text{centj})
{
 f\theta_1 = \theta; \]
\[ r = (\text{centi-i})*(\text{centi-i}) + (\text{centj-j})*(\text{centj-j}); \]
\[ r = \sqrt{r}; \]
\[ \text{newj} = \text{centj} + r*\cos(\phi_1) + 0.5; \]
\[ \text{newi} = \text{centi} + r*\sin(\phi_1) + 0.5; \]
\[ \text{new[newi][newj]} = 0; \]
}
(centj-j)*(centj-j);
    r = sqrt(r);

newj = centj + r*cos(ftheta) + 0.5;
newi = centi - r*sin(ftheta) + 0.5;
new[newi][newj] = 0;
}
else if(i==centi && j<centj)
{
    ftheta = theta;
    r = (centj-j)*(centj-j);
    r = sqrt(r);

newj = centj - r*cos(ftheta) + 0.5;
newi = centi - r*sin(ftheta) + 0.5;
new[newi][newj] = 0;
}
else if(i>centi && j==centj)
{
    ftheta = theta;
    r = (centj-j)*(centj-j);
    r = sqrt(r);

newj = centj - r*cos(ftheta) + 0.5;
newi = centi + r*sin(ftheta) + 0.5;
new[newi][newj] = 0;
}
else if(i==centi && j==centj)
{
    new[centi][centj] = 0;
}
}

fpout = fopen(name2,"w");
fwrite(new,512*512,1,fpout);
fclose(fpout);
/*THIS IS THE PROGRAM TO CALCULATE FAST HADAMARD TRANSFORM*/
/* USING UNIPROCESSOR */
/* SOFTWARE DEVELOPED BY: */
/* SAIFUDDIN MOIZ */
/************************************************************************/

#include <stdio.h>
#include <math.h>

char image[512][512],image3[512][512];
long image2[262144],image1[262144];

main()
{

    /* assignment of variable */

    int i,j,k,m,l,
        int n,nv2,nm1,le,le1,ip,temp,
        int n2,n4,ix,iy,lm,lml,
        long x,y;
    float Scale_fac;
    double pow();
    double log10();
    double t,s,d,ln,f,a;
    char name[30];
    char namel[30];
    FILE *inptr;
    FILE *ouptr;

    printf("Fast Hadamard Transform using Uniprocessor\n\n");
    printf("Enter the order of the image:");
    scanf("%d",&l);

    printf("Enter the input filename:");
    scanf("%s",name);

    printf("Enter the output filename:");
    scanf("%s",namel);

    inptr = fopen(name,"r");
        if ( inptr == (FILE*)NULL )
            {
                printf("Cannot open %s\n", name);
                exit(1);
            }
    printf("\n");
    /* read the image */
    printf("Reading the image...\n\n");
fread(image, 512*512, 1, inptr);
k = 0;
for(i=0; i<=l-1; i++)
{
    for(j=0; j<=l-1; j++)
    {
        /* fscanf(inptr, "%3d", &image[i][j]);
        printf("%3d", image[i][j]);*/
        imagel[k] = (long)image[i][j];
        k++;
    }
}
printf("Done with reading...\n\n");

printf("Calculating the 2-D FHT.....\n\n");
printf("Please wait.............\n\n");

t=2.0;
d=l*l;
s = log10(d)/log10(t)+(1E-10);
ln = (int)s;
a = 2;
n = pow(a,ln);
nv2 = n/2;
ml = n-1;
j = 1;
for(i=1; i<=ml; i++)
{
    if(i<j)
    {
        temp = imagel[j];
        imagel[j] = imagel[i];
        imagel[i] = temp;
    }
    k = nv2;
    while(k<j)
    {
        j = j-k;
        k = k/2;
    }
    j = j+k;
}
for(f=1; f<=ln; f++)
{
    le = pow(a,f);
    lel = le/2;
    for(j=1; j<=lel; j++)
    {
        for(i=j; i<=n; i+=le)
        {
            ip = i+le1;
temp = imagel[ip];
imagel[ip] = abs(imagel[i]-temp);
imagel[i] = imagel[i]+temp;
}
}
n4 = n/4; n2 = n/2;
for(i=1; i<=ln-1; i++)
{
    lm = i-1;
    lml = pow(a,lm);
    for(m=1; m<=lml; m++)
    {
        for(j=1; j<=n4; j++)
        {
            iy = n2+j+(m-1)*n/lml;
            ix = iy+n4;
            temp = imagel[ix];
            imagel[ix] = imagel[iy];
            imagel[iy] = temp;
        }
    }n2 = n2/4; n4 = n4/4;
}

/*x = image1[0];
y = 127.0;
Scale_fac = y/(float)x;
for(i=0; i<=(l*l); i++)
{
    if(image1[i]<0)
        image1[i] = (-1)*(image1[i]);
    image2[i] = (double)image1[i]*Scale_fac;
}*/x = image1[0];
for(i=0; i<=l*l; i++)
image2[i] = (double)image1[i]/(float)x;

k = 0;
for(i=0; i<=l-1; i++)
{
    for(j=0; j<=l-1; j++)
    {
        image3[i][j] = (char)image2[k];
k++;
    }
}

outptr = fopen(name1,"w");
fwrite(image1,512*512,1,outptr);
/*printf(" The transformed image is as follows\n");
k=1;
for(i=1; i<=l; i++)
{
    for(j=1; j<=l; j++)
    {
        printf("%6d", image2[k]);
        ++k;
    }
    printf("\n");
}*/
fclose(inptr);
fclose(ouptr);
/* PROGRAM TO FIND THE FAST HADAMARD TRANSFORM */
/* USING PARALLEL PROCESSOR - PARENT PROGRAM */
/* SOFTWARE DEVELOPED BY: */
/* SAIFUDDIN MOIZ */

#include <stdio.h>
#include <math.h>
#include "ringp.h"
static char image[512][512],image1[512][512];
user-main()
{
  /* assignment of variable */
  long i,j,k,f,l,sub,array[16],arrav[6];
  long ntermsn,x,m;
  long nterms;
  char name[30];

  FILE *inptr;
  FILE *ouptr;

  printf("\n 2-D FHT using ring topology\n\n");
  printf("Enter the size of image: ");
  scanf("%d", \&l);
  printf("Enter the input filename: ");
  scanf("%s", name);

  inptr = fopen( name,"r" );
  if ( inptr == (FILE *)NULL )
  {
    printf("Cannot open %s\n", name);
    exit(1);
  }

  /* read the image */
  printf(" Reading the image....\n");
  fread ( image, 512*512, 1, inptr );
  fclose(inptr);

  printf("\n");
  for(i=0; i<=l-1; i++)
  {
    for(j=0; j<=l-1; j++)
    {
      fscanf(inptr, "%3d", \&image[i][j]);
      printf("%3d", image[i][j]);
    }
  
  printf("\n");
ring(SAME, "child.exe");
nterms = 16384;
dnldac(image, nterms);
printf("The image has been downloaded\n");

upldac(image1,nterms);
printf("The final image has been received\n");

ouptr = fopen("image1", "w");
fwrite(image1, 512*512, 1, ouptr);
fclose(ouptr);

/*printf("Second image has been received from each child\n");
printf("The new image is as follows\n");
printf("\n");
f=0;
for(j=0; j<=l-1; j++)
{
    for(i=0; i<=l-1; i++)
    {
        printf("%4d", image1[i][j]);
    }
    printf("\n");
}
printf("\n");*/

/* dnldai(image2, nterms);
printf(" Third image has been downloaded\n");
upldai(image4,nterms);*/
 /*printf(" Fourth image has been received from each child\n");
printf(" The final image is as follows\n");
printf("\n");
k=0;
for( i=0; i<=l-1; i++)
{
    for( j=0; j<=l-1; j++)
    {
        printf("%5d", image4[k]);
        ++k;
    }
    printf("\n");
}
printf("\n");*/
}
# include "ringc.h"
char array[16384];
user_main()
{
    /* assignment of variable */
    long *imagel,*image2;
    long *buff,*buff1,*result;
    long nterms,ntermsn;
    long sender,i,l;
    long k,p1,r,j;

    l = 512;
    nterms = 16384;
    dnldac( array,nterms);

    mark();
    imagel= (long *)bmalloc(16384*sizeof(long));

    mark();
    image2= (long *)bmalloc(16384*sizeof(long));

    for(i=0; i<=16383; i++)
        imagel[i] = (long)array[i];

    for(i=0; i<=16383; i++)
        image2[i]=0;

    for(i=0; i<=16384; i+=1)
    {
        Transf(image1+i, image2+i,l);
    }

    /* upldai(image2, nterms);*/

    mark();
    buff = (long*)bmalloc(16384*sizeof(long));
    for(i=0; i<=16383; i++)
        buff[i]=0;
mark();
buffl = (long*) bmalloc(16384*sizeof(long));
for(i=0; i<=16383; i++)
buffl[i]=0;

EMPLATE THE USING THE
INTER -NODE COMMUNICATION
ntermsn = 8192;
for (sender = 0; sender<=15; sender++)
{
for (i = 0; i<=16383; i++)
buff[i] = image2[i];
brdcstal(buff, ntermsn, sender);
brdcstal(&buff[8192], ntermsn, sender);
}
k=32*childnum;
for(i=0;i<=31;i++)
{
pl=32*sender;
j=0;
for(r=pl;r<=pl+31;r++)
{
buffl[r+512*i]=buff[k+512*j];
j++;
}
k++;
}
mark();
result = (long*) bmalloc(16384*sizeof(long));
for(i=0; i<=16383; i++)
result[i]=0;

/****** TAKING THE TRANSPOSE AND APPLYING THE 1D TRANSFORMATION TO GET THE FINAL RESULTS******/
for(i=0; i<=16384; i+=1)
{
Transf(buff1+i, result+i,1);
}
for(i=0; i<=16383; i++)
array[i] = ((result[i]+130560)/(2*130560)) * 255;
upldac(array, nterms);
mfree(result);
mfree(buff1);
mfree(buff);
mfree(image2);
mfree(image1);

/* */
/*
FUNCTION FOR 1-D FHT
*/
/* */

long *pa, *pb;
long l;
{
long nl, shiftl, r, ima, w, x, u, pl;
long *pc;
double pow();
double a, s, shift, t, d, q;
double log10();
t = 2.0, nl = 0;
d = 1;
s = log10(d) / log10(t) + (1E-10);
r = (int)s;

for (u = 0; u <= l-1; u++)
{
pl = 0; ima = 0;
pc = pa;
for (x = 0; x <= l-1; x++)
{
a = 2;
w = 0;
for (q = 0; q <= r-1; q++)
{
    shift = pow(a, q);
    shiftl = (int) shift;
    if(( shiftl & x) != 0)
    {
        if(( shiftl & u) != 0)
            nl++;
    }
}
w = nl;
nl = 0;
if (w == 0)
    ima = *pc++;
else if ((w % 2) == 0)
    ima = *pc++;
else
    ima = (*pc++) * (-1);
    pl = pl + ima;
}
PARALLEL LABELLING ALGORITHM
PARENT PROGRAM.

#include <stdio.h>
#include "ringp.h"
char image[512][512];

user_main()
{
    int c[16][10], k;
    int i, j, ntermsn, hope[16];
    char name1[25], name2[25], name3[25], name4[25];
    char signal;
    double tsec;
    FILE *fp;

    printf("Enter the name of file to be opened :\n");
    scanf("%s",name1);
    printf("The input file name is given by : %s\n",name1);
    /*
    printf("Enter the name of file to be stored :\n");
    scanf("%s",name2);
    printf("The output file name is given by : %s\n",name2);
    /*
    printf("Enter the name of second file to be stored :\n");
    scanf("%s",name3);
    printf("The output file name is given by : %s\n",name3);
    */
    printf("Enter the name of second file to be stored :\n");
    scanf("%s",name4);
    printf("The output file name is given by : %s\n",name4);

    fp = fopen(name1,"r");
    fread(image, 512*512, 1, fp);
    fclose(fp);
    ntermsn = 32*512;
    ring(SAME,"newc.exe");
    dnldac(image, ntermsn);
c_rcvc( signal );
zero_timer( 0 );

c_rcvc( signal );
read_timer( &tsec );

printf("The parallel time is given by %f
seconds\n",tsec);

upldac(image, ntermssn);

fp = fopen(name4,"w");
fwrite(image, 512*512, 1, fp);
fclose(fp);
```c
#include "ringc.h"
#define DIM 10000
char image[34][512];
int i_arr[ DIM], j_arr[ DIM];

user_main()
{
    int i, j, ntermsn;
    int k, a, b, store, check;
    int n, filler, hope = 10, signal, delay;
    int il, jl, new_fill, val, compare, c[10];
    char sign;

    sign = 10;

    ntermsn = 32*512;
    dnldac(&image[1][0], ntermsn);

    for(j=0; j<=511; j++)
    {
        image[0][j] = 0;
        image[33][j] = 0;
    }

    if(childnum == 0)
    {
        c_sndc( sign );
    }

    for(n=1; n<=5; n++)
    {
        check = 0;
        store = 0;

        for(i=0; i<DIM; i++)
        {
            i_arr[i] = 0;
            j_arr[i] = 0;
        }

        for(i=1; i<=32; i++)
        {
```
for(j=0; j<=511; j++)
{
    if( image[i][j] == 60 )
    {
        check = 1;
        i_arrry[0] = i;
        j_arrry[0] = j;
    }
    if( check == 1)
    {
        i = 512;
        j = 512;
    }
}

if(check == 1)
filler = 6*childnum + n;
else
filler = 0;

for(k=0; k<DIM; k++)
{
    if(filler == 0)
    {
        a = 1;
        b = 1;
    } else
    {
        a = i_arrry[k];
        b = j_arrry[k];
    }

    image[a][b] = filler;

    if( image[a][b+1] == 60 )
    {
        image[a][b+1] = filler;
        i_arrry[store] = a;
        j_arrry[store] = b+1;
        store++;
    }

    if( image[a+1][b+1] == 60 )
    {
        image[a+1][b+1] = filler;
        i_arrry[store] = a+1;
        }
if( image[a+1][b] == 60 )
{
    image[a+1][b] = filler;
    i_arr[store] = a+1;
    j_arr[store] = b;
    store++;
}

if( image[a][b-1] == 60 )
{
    image[a][b-1] = filler;
    i_arr[store] = a;
    j_arr[store] = b-1;
    store++;
}

if( image[a-1][b-1] == 60 )
{
    image[a-1][b-1] = filler;
    i_arr[store] = a-1;
    j_arr[store] = b-1;
    store++;
}

if( image[a-1][b] == 60 )
{
    image[a-1][b] = filler;
    i_arr[store] = a-1;
    j_arr[store] = b;
    store++;
}

/* upldac(&image[1][0], ntermsn); */

for(signal=0;signal<=14;signal++)
{
    val = 0;

    if(childnum == signal)
    {
        sndac(&image[32][0],512,RIGHT);
    }
if(childnum == signal+1) {
    rcvac(&image[0][0], 512, LEFT);
}

if(childnum == signal+1) {

    for(i=0; i<=9; i++)
        c[i] = 0;
    found = 0;
    for(j=1; j<=511; j++)
    {
        if(image[0][j-1] != 0 && image[0][j] == 0)
        {
            c[found] = j-1;
            found++;
        }
    }

    for(i=0; i<=9; i++)
    {
        if(c[i] != 0)
        {
            val = c[i];
            for(il=val-10; il<=val+10; il++)
            {
                if(image[1][il] != 0)
                {
                    new_fill = image[0][val];
                    compare = image[1][il];
                    il = val+12;
                }
            }
            for(il=1; il<=32; il++)
            {
                for(jl=0; jl<=511; jl++)
                {
                    if(image[1][jl] == compare)
                        image[1][jl] = new_fill;
                }
            }
        }
    }
}
signal = 15;
while(signal != 0)
{
    val = 0;

    if(childnum == signal)
    {
        sndac(&image[1][0], 512, LEFT);
    }
    if(childnum == signal-1)
    {
        rcvac(&image[33][0], 512, RIGHT);
    }

    if(childnum == signal-1)
    {
        for(i=0; i<=9; i++)
        {
            c[i] = 0;
            found = 0;
        }
        for(j=1; j<=511; j++)
        {
            if(image[33][j-1] != 0 && image[33][j] == 0)
            {
                c[found] = j-1;
                found++;
            }
        }
    }
    for(i=0; i<=9; i++)
    {
        if(c[i] != 0)
        {
            val = c[i];
            for(i1=val-10; i1<=val+10; i1++)
{  if(image[32][il] != 0)
  {
    new_fill =
    compare = image[32][il];
    il = val+12;
  }
}
for(il=1;il<=32;il++)
{
  for(jl=0;jl<=511;jl++)
  {
    if(image[il][jl] == compare)
      image[il][jl] =
    new_fill;
  }
}
signal = signal-1;
}
if(childnum == 0)
{
  c_sndc( sign );
}
upldac(&image[1][0], ntermsn);
}
/* PROGRAM TO CLASSIFY OBJECTS USING THEIR HADAMARD DOMAIN FEATURES */

#include <stdio.h>
#include <math.h>

main()
{
    int i, j;
    double samp1[16][10], samp2[16][10], samp3[16][10];
    double temp[3][5];
    double prto1[16][10], prto2[16][10], prto3[16][10];
    double a[3], b[3], c[3];
    int samples;

    FILE *fp1, *fp2, *fp3;

    /* the three original files are read in here */
    fp1 = fopen("samp1.dat","r");
    fread(samp1,8,16*10,fp1);
    fclose(fp1);

    fp2 = fopen("samp2.dat","r");
    fread(samp2,8,16*10,fp2);
    fclose(fp2);

    fp3 = fopen("samp3.dat","r");
    fread(samp3,8,16*10,fp3);
    fclose(fp3);

    /* the three prototype files are read in here */
    fp1 = fopen("prto1.dat","r");
    fread(prto1,8,16*10,fp1);
    fclose(fp1);

    fp2 = fopen("prto2.dat","r");
    fread(prto2,8,16*10,fp2);
    fclose(fp2);

    fp3 = fopen("prto3.dat","r");
    fread(prto3,8,16*10,fp3);
    fclose(fp3);

    /* the number of objects is read in here */
    printf("Enter the number of samples to be used in the classification \
\n");
    scanf("%d", &samples);
printf("the number of samples is : %d \n\n", samples);

/* the minimum distance is calculated below */

/* for the first prototype */

for(i=0; i<samples; i++)
{
    temp[0][i] = samp1[i][0] - proto1[i][0];
    temp[1][i] = samp2[i][0] - proto1[i][0];
    temp[2][i] = samp3[i][0] - proto1[i][0];
}

for(j=0; j<samples; j++)
{
    a[0]+ = temp[0][j] * temp[0][j];
    b[0]+ = temp[1][j] * temp[1][j];
    c[0]+ = temp[2][j] * temp[2][j];
}

a[0] = sqrt(a[0]);
b[0] = sqrt(b[0]);
c[0] = sqrt(c[0]);

/* for the second prototype */

for(i=0; i<samples; i++)
{
    temp[0][i] = samp1[i][0] - proto2[i][0];
    temp[1][i] = samp2[i][0] - proto2[i][0];
    temp[2][i] = samp3[i][0] - proto2[i][0];
}

for(j=0; j<samples; j++)
{
    a[1]+ = temp[0][j] * temp[0][j];
    b[1]+ = temp[1][j] * temp[1][j];
}

a[1] = sqrt(a[1]);
b[1] = sqrt(b[1]);
c[1] = sqrt(c[1]);

/* for the third prototype */

for(i=0; i<samples; i++)
{
    temp[0][i] = samp1[i][0] - proto3[i][0];
    temp[1][i] = samp2[i][0] - proto3[i][0];
    temp[2][i] = samp3[i][0] - proto3[i][0];
}

for(j=0; j<samples; j++)
{
{ 
    a[2] += temp[0][j] * temp[0][j];
    b[2] += temp[1][j] * temp[1][j];
}

a[2] = sqrt(a[2]);
b[2] = sqrt(b[2]);
c[2] = sqrt(c[2]);

for(i=0;i<4;i++)
{
    printf("a[%d] = %f\tb[%d] = %f\tc[%d] = %f\n", i, a[i], i, b[i], i, c[i]);
}

}
```c
#include <stdio.h>
#include <math.h>

main()
{
    static int a1[32][32], a2[32][32], a3[32][32];
    int d12[10], d13[10], d23[10];
    int d0_12[10], d0_13[10], d0_23[10];
    int d1_12[10], d1_13[10], d1_23[10];
    int d2_12[10], d2_13[10], d2_23[10];
    int d3_12[10], d3_13[10], d3_23[10];

double x0_12, x0_13, x0_23;
double x1_12, x1_13, x1_23;
double x2_12, x2_13, x2_23;
double x3_12, x3_13, x3_23;
double sqrt();
    int i, j;

    char name1[25], name2[25], name3[25];
    FILE *inptr1;
    FILE *inptr2;
    FILE *inptr3;

    printf("\n Enter the first object coefficient file:");
    scanf("%s", name1);

    printf("\n Enter the Second object coefficient file:");
    scanf("%s", name2);

    printf("\n Enter the Third object coefficient file:");
    scanf("%s", name3);

    inptr1 = fopen(name1,"r");
inptr2 = fopen(name2,"r");
inptr3 = fopen(name3,"r");

    for(i=0; i<=31; i++)
    {
        for(j=0; j<=31; j++)
        {
            fscanf(inptr1, "%3d", &a1[i][j]);
            fscanf(inptr2, "%3d", &a2[i][j]);
            fscanf(inptr3, "%3d", &a3[i][j]);
        }
    }
```
printf("Please wait for processing...........
");

for(i=1; i<=7; i++)
{
x0_12 = (a1[i][i]-a2[i][i]);
x1_12 = (a1[i-1][i]-a2[i-1][i]);
x2_12 = (a1[i-1][i+1]-a2[i-1][i+1]);
x3_12 = (a1[i][i-1]-a2[i][i-1]);
d0_12[i] = sqrt(x0_12*x0_12);
d1_12[i] = sqrt(x1_12*x1_12);
d2_12[i] = sqrt(x2_12*x2_12);
d3_12[i] = sqrt(x3_12*x3_12);
d12[i] = sqrt(x0_12*x0_12 + x1_12*x1_12 + x2_12*x2_12 + x3_12*x3_12);

x0_13 = (a1[i][i]-a3[i][i]);
x1_13 = (a1[i-1][i]-a3[i-1][i]);
x2_13 = (a1[i-1][i+1]-a3[i-1][i+1]);
x3_13 = (a1[i][i-1]-a3[i][i-1]);
d0_13[i] = sqrt(x0_13*x0_13);
d1_13[i] = sqrt(x1_13*x1_13);
d2_13[i] = sqrt(x2_13*x2_13);
d3_13[i] = sqrt(x3_13*x3_13);
d13[i] = sqrt(x0_13*x0_13 + x1_13*x1_13 + x2_13*x2_13 + x3_13*x3_13);

x0_23 = (a2[i][i]-a3[i][i]);
x1_23 = (a2[i-1][i]-a3[i-1][i]);
x2_23 = (a2[i-1][i+1]-a3[i-1][i+1]);
x3_23 = (a2[i][i-1]-a3[i][i-1]);
d0_23[i] = sqrt(x0_23*x0_23);
d1_23[i] = sqrt(x1_23*x1_23);
d2_23[i] = sqrt(x2_23*x2_23);
d3_23[i] = sqrt(x3_23*x3_23);
d23[i] = sqrt(x0_23*x0_23 + x1_23*x1_23 + x2_23*x2_23 + x3_23*x3_23);
}

printf("The class distance are as follows....\n");
printf("Coefficient  d12  13  d23\n");

for(i=1; i<=7; i++)
{
printf("H[%d][%d] %3d %3d %3d %3d %3d %3d
%3d %3d %3d %3d %3d %3d %3d %3d %3d
", i, i, d0_12[i], d1_12[i], d2_12[i], d3_12[i],
d12[i], d0_13[i], d1_13[i], d2_13[i], d3_13[i],
d13[i], d0_23[i], d1_23[i], d2_23[i], d3_23[i], d23[i]);
}
fclose(inptr1);
fclose(inptr2);
fclose(inptr3);