ABSTRACT

In this paper we describe an SIMD parallel algorithm for efficient implementation of template matching for object recognition. The correlation coefficient is used to find the best match between the template of an object and the given search image. A fast computing algorithm is developed and implemented on a 16-node hypercube concurrent system.

I. INTRODUCTION

Template matching by image correlation has been used in many areas of digital image processing. This process involves finding the best match between a template image of an object and a search image. These two images are compared to determine whether or not the search image contains the object in the template [1]. Although this computation takes extremely long time in sequential implementation, parallel algorithms can reduce the processing time to an acceptable level [2]. Here we describe a parallel implementation technique for template matching on a hypercube system (Ametek S-14) which has a smaller number of processing elements (N=16) than the size of the search image (512x512).

II. IMPLEMENTATION OF TEMPLATE MATCHING

Template matching is performed using cross correlation which is given by

\[ C(m,n) = \sum_{i} \sum_{j} P(i,j) T(i-m,j-n) \] (1)

where \( P(i,j) \) is an \( N \times N \) search image and \( T(i,j) \) is an \( M \times M \) template image (see Fig.1). The template is moved over the search area \( P(i,j) \) and the subarea with the maximum value of correlation in the search image is selected as the one similar to the template. Since \( C(m,n) \) is dependent on intensity, we use the correlation coefficient [1] which is given by

\[ C_r(m,n) = \frac{\sum (P(i,j)-p(i,j))(T(i-m,j-n)-t)}{\sqrt{\sum (P(i,j)-p(i,j))^2 \sum (T(i-m,j-n)-t)^2}} \] (2)

where \( p(i,j) \) is the average value of the subimage enclosed by the template \( T(i,j) \), and \( t \) is the average value of the window \( T(i,j) \). In computing Eq.2, the summations are taken over the domain where the template \( T(i,j) \) is defined. Here \( C_r(m,n) \) is simplified to

\[ C_r(m,n) = \frac{\sum (P(i,j)-p(i,j)) T(i-m,j-n)}{s^2} \]

where \( s = \sqrt{\sum (P(i,j)-p(i,j))^2} \), \( m \leq i \leq M + m \), \( n \leq j \leq M+n \).

Figure 2 shows a flow-diagram of the computation of correlation coefficient. The computation time for Eq.3 is further reduced by eliminating some repeated processes on the same data. Here we describe a method for computing the stated summations based on the one developed in [2]. However, the proposed scheme requires considerably less memory space.

Figure 3 shows the correlation process. The matching procedure starts from the upper left hand area of the input image. Let \( q(m,n) \) represent the sum of a local image area whose upper left hand corner pixel is at \( (m,n) \). Thus

\[ q(m,n) = q(m,n-1) + \sum_{i} [P(i,M+n-1)-P(i,n-1)] \]

where \( n > 0 \) and \( m > 0 \). Let a 2-element array \( sum(r) \) be a temporary storage. The location \( sum(0) \) contains the sum of the leftmost subimage. Every time the template moves to the leftmost position of the search image, the sum of its upper neighbor is provided by \( sum(0) \) for computation. The value contained in \( sum(0) \) is then updated by the new result for the computation of the next subimage. The memory location \( sum(1) \) contains the sum of the left adjacent subimage of the current area. It is then updated by the present value for the computation of the next subimage as

\[ q(m,n) = sum(1) + \sum_{i} [P(i,M+n-1)-P(i,n-1)] \]

where \( n > 0 \) and \( m > 0 \). The advantage of this method is that the required memory space is
reduced from an \((N+M-1) \times (N+M-1)\) array to a 2-element array by storing only the summation result of the previous subimage.

III. HYPERCUBE IMPLEMENTATION

An n-dimensional hypercube machine consists of \(2^n\) processors arranged in an n-D hypercube space [3]. Nodes are placed at corners of an n-D cube defined in the n-D space. All nodes are numbered in n-bit Gray code [3]. Mesh and ring are two commonly used topologies which can be easily mapped onto hypercube computers. Figures 4 and 5 show a 4x4 mesh and a 16-node ring topology mapped onto a 16-node hypercube.

In Figures 6 and 7, an input image of size 512x512 is equally divided into sixteen 128x128 and 32x512 subimages for mesh and ring implementations of the template matching algorithms, respectively. Every subimage is downloaded by the host computer to a corresponding PE. An NxM template is broadcast to all nodes so that every PE can compute correlation coefficient independently until the template reaches the subimage boundaries. To perform template matching along the subimage boundaries, the neighboring nodes of a particular PE compute \(C(x,y)\) for the image portion that they contain. The overall result in a node can be completed by combining all the intermediate results obtained by its neighbors. We let all nodes receive the boundary data from their neighbors before starting any computations. After all the nodes receive their necessary data, they perform template matching without any interprocessor communication.

The performance of parallel implementations of an algorithm can be measured by the speedup, \(S\), which is defined as \(S = \frac{T_S}{T_P}\) where \(T_S\) is the total execution time for the sequential implementation in a single node of the concurrent system and \(T_P\) : the total execution time for the parallel implementation. The ideal speedup ignoring the inter-node communications is given by

\[
S_{id} = \frac{(N-A)^2}{(N-k^2)(N-k^2)} t_u \quad (N-A) \leq k
\]

(6)

Here \((N-A)^2\) is the number of times that the template moves for computing correlation in a single node of the concurrent system, \(N^2/k\) is the number of times that the template moves for computing correlation in a parallel computer, \(t_u\) is the computation time for a match area, and \(A=M-1\) (see Figures 6 and 8). However, in real-time implementation, we also consider the communication between nodes since it affects the speedup considerably. Thus the speedup for mesh \((S_m)\) and ring \((S_r)\) implementations are given by

\[
S_m = \frac{(N-A)^2}{(N^2/k) t_u + 2A(A+M/k)} + t_{on} \quad \text{(7)}
\]

\[
S_r = \frac{(N-A)^2}{(N^2/k) t_u + (NA) t_o + t_{on}} \quad \text{(8)}
\]

where \(t_{on}\) is the time taken to transfer one byte of data, \(t_{on}\) is the system overhead time, and \(a\) is the number of data transfer among nodes.

Plots of the speedup versus different parameters are illustrated in Fig.9 by using the typical values of \(t_u\) and \(t_{on}\) for Ametek S-14 (10 and 300 microseconds).

IV. EXPERIMENTAL RESULTS

The described parallel image correlation method has been implemented on the Ametek S-14 concurrent hypercube system hosted by the VAX 11/750 minicomputer. Image acquisition has been performed through an RCA camera and the VICOM image processor. The parent and the child programs were written in C programming language. The parent program runs on the VAX and the child program is executed in the nodes of S-14.

The experiment shown in Fig.10 involves matching an English character. The 512x512 input image contains some characters, and the template of size 26x20 includes the character E. Using the algorithm developed, E is located and a bounding box is placed around it for identification. Fig.11 shows the other experiment which uses a 128x200 template to detect the scissors. The bounding box enclosing the object indicates that the desired object is correctly located.

V. CONCLUSIONS

In this paper, we have described an efficient implementation of the correlation-based image template matching. The overall execution time is significantly reduced by simplifying the computational complexity of correlation coefficient and effectively parallelizing the matching procedure. Equations of speedup factor are derived to evaluate the performance of this algorithm for the mesh and ring topology implementations.

REFERENCES


Figure 1

Figure 2

Figure 3

Figure 4