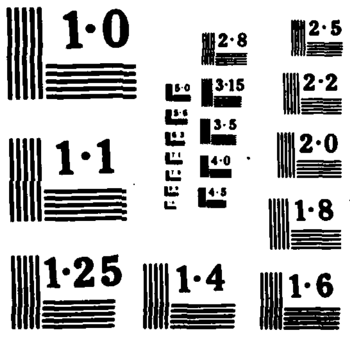
1·0
2·8
2·5
3·15
2·2
3·5
2·0
4·0
1·1
4·5
1·8
1·25
1·4
1·6

(12)

Template Matching on Parallel Architectures

David Sher
Computer Science Department
The University of Rochester
Rochester, New York 14627

TR 156
July 1985

N00014-82-K-0193

Contract N00014-82-K-0193

rochester

Department of Computer Science
University of Rochester
Rochester, New York 14627

12

Template Matching on Parallel Architectures

David Sher
Computer Science Department
The University of Rochester
Rochester, New York 14627

TR 156
July 1985

N00014-82-K-0193

Abstract

Many important problems in computer vision can be characterized as template matching problems on edge images. Some examples are circle detection and line detection. Two techniques for template matching are the Hough transform and correlation. There are two algorithms for correlation: a shift and add based technique and a Fourier transform based technique. The most efficient algorithm of these three varies depending on the size of the template and the structure of the image. On different parallel architectures the choice of algorithms for a specific problem is different. This paper describes two parallel architectures : the WARP and the Butterfly and describes why and how the criterion for making the choice of algorithms differs between the two machines.

This material is based upon work supported under a National Science Foundation Graduate Fellowship grant number SPE-8350104. This work was supported in part by NSF Grant RCD-8450125 and a Defense Advanced Research Project Authority grant number N0014-82-K-0193

0

1. Introduction

In this paper the problem of template matching between edge images is explored. Edge images are the thresholded output of running an edge detector over a digitized image. The resultant data structure is an array of boolean data describing where the edges are in the image and an array of orientations for the edges. A template is a representation of a small structure in a similar format to that of the image. An example of such a template is shown in figure 1. The process of template matching on edge images finds the places in the image that are similar to the template.

This paper discusses the implementation of template matching on multiprocessors. The two multiprocessors that are examined in this paper are the WARP and the Butterfly. The WARP is a heavily pipelined multiprocessor. According to the system promulgated by Flynn for categorizing multiprocessors it is a MISD architecture [5]. A MISD architecture is a multiple instruction stream single data stream machine. The WARP is MISD because each of its processors can run different code simultaneously and there is one stream through which data can be piped.

The WARP is a pipeline of 10 independently programmable synchronized processors. Each processor uses pipelined functional units that can be accessed in parallel.

The other multiprocessor that is examined is the Butterfly. The Butterfly contains 128 68000 based processors. Each processor has its own local memory. Each processor executes instructions out of its local memory. The processors run asynchronously. Thus according to Flynn's categories the Butterfly is a MIMD machine.

The processors of the Butterfly are connected with a Fourier Transform or Butterfly switching network. Through this network they can access the other processors' local memory. On the Butterfly a remote read through the switch takes 6 times the time of a local memory reference. A write takes twice the time of a local memory reference. The processors and the switch run asynchronously.

In this paper I will show that for certain kinds of templates the most efficient algorithms for matching on the WARP and the Butterfly are radically different. I will also examine a generalization of these two architectures to examine the importance of the speed of floating point operations.

X

ltr per

Figure 1: Template of A Diamond Pattern

-	-	$\frac{7\pi}{4}$	$\frac{\pi}{4}$	-	-
-	$\frac{7\pi}{4}$	-	-	$\frac{\pi}{4}$	-
$\frac{7\pi}{4}$	-	-	-	-	$\frac{\pi}{4}$
$\frac{5\pi}{4}$	-	-	-	-	$\frac{3\pi}{4}$
-	$\frac{5\pi}{4}$	-	-	$\frac{3\pi}{4}$	-
-	-	$\frac{5\pi}{4}$	$\frac{3\pi}{4}$	-	-

A-1

2. Descriptions of the WARP and Butterfly

2.1. The WARP Architecture

The WARP is a machine currently being fabricated at CMU by the VLSI group there [6] [7]. The WARP is an array of 10 processors arranged in a pipeline. Each processor can only communicate with the previous and next processor in the pipeline. There is a host processor that sends information to the first processor and receives the results from the last. There is an interface that takes integer data from the host and translates it to floating point for the WARP and vice-versa. Currently the exact machine to be used as the WARP's host is undetermined. A detailed view of the WARP's processors and a description of a host that would be ideal for the WARP for image processing is in this section.

2.1.1. The WARP's Processors

The WARP's cycle time is 200ns. Each of the WARP's processors is heavily pipelined. The pipelining makes it difficult to program the WARP and analyze its behavior. If one considers the WARP as a machine with a 1000 ns cycle time one can ignore much of the pipelining. In this section I describe the WARP's processors in these terms.

In a 1000ns cycle each processor can retrieve 10 32 bit floating point numbers from either its predecessor or its successor or itself. It can send 10 numbers in that same cycle. It can in the same cycle do 5 memory reads and 5 memory writes. An indirect access counts as 1.5 memory accesses. In the same cycle it can send 10 numbers to the floating point multiplier from any part of the machine. It can also send 10 numbers to its floating point alu. The multiplier and alu can each do 5 operations in each cycle. The results of the computations set up in a cycle are available in the second cycle thereafter. The alu can compute: fix (float -> 24 bit integer) float (24 bit integer -> float) $a+b$ $a-b$ $b-a$ $|a|+|b|$ $|a-b|$ $|a+b|$

Both the alu and the multiplier can compute $\frac{1}{x}$ and $\frac{1}{\sqrt{x}}$ (because of an accident of the design). The warp can test for 0 or negative on the output of the ALU and behave conditionally thereupon. It can generate and read two flag bits that it can send or read to the next WARP along.

Each WARP processor has 2K of 32 bit word memory for storing data and tables.

This is a simplified description of the WARP. For a more detailed description see [6].

2.1.2. The Ideal Host: the WIMP

WIMP stands for Wide Integer Memory Processor. Experience programming the WARP suggests that a good host for the WARP should have these properties:

(1) It should be synchronous with the WARP

(2) It should have i/o bandwidth within an order of magnitude of the WARP's.

(3) It should have a large memory to store large intermediate data structures.

(4) It should be able to do simple integer operations and memory references at a speed comparable to the WARP's floating point operation speed.

A machine that meets these specifications is within the capability of the currently available hardware. The rest of this section gives a more detailed description of an architecture that is plausible and fulfills these requirements.

2.1.2.1. The Memory

The WIMP should be an integer processor with a 24 bit word size to be compatible with the WARP. The address space pointed to by a 24 bit word is 16 megawords. A 16 megaword address space allows it to store 2 1K by 1K images with 8 words of information per pixel.

2.1.2.2. The Processor

Like the WARP the WIMP should be able to use all its functional units in parallel. The WIMP will have 5 200ns cycles in one of the 1000ns WARP cycles described above. During one 200 ns cycle the WIMP should be able to:

(1) Read one 24bit integer from the WARP

(2) Write one 24bit integer to the WARP

(3) Read one 24bit integer from main memory

(4) Write one 24bit integer to main memory

(5) Write one 24bit integer to the register file

(6) Read one 24bit integer to the register file

(7) Apply one alu operation

(8) Apply one multiplication

The register file should have at least 16 24bit registers.

2.2. The Butterfly Architecture

The Butterfly is a network of 128 tightly coupled asynchronous 68000's. Each 68000 also has associated with it a memory management processor that handles memory requests. It also manages other aspects of memory management and does some simple functions like block move. Because the processors are not arranged in a full crossbar there is the possibility of two remote requests conflicting. It has been found that for many regular communication graphs including arrays these conflicts can be eliminated [14]. A processor will block until a remote memory reference is finished.

A 68000 local memory reference (through the built in virtual memory system) takes 625ns. A remote read through the switch takes about 3750 ns and a write takes about 1250 ns. Block moves of data amortize some of the handshaking overhead over the entire transaction thus can run faster than the equivalent set of single remote memory references [15]

There is no floating point hardware or microcode on the Butterfly. The floating point operations are done in software. The arithmetic capabilities are just those of the 68000 [12].

A program has been written that tests the speed of various operations on a Butterfly. The results can be considered to be accurate to within 30%. These timings suffice for the purposes of this paper. The results are:

Timings of Operations on the Butterfly

Operation (memory to memory)	Speed (in microseconds)
integer assignment	4.30
floating point assignment	11.29
integer addition	4.15
floating point addition	295.59
integer multiplication	68.24
floating point multiplication	474.76

The table above shows that a floating point multiply takes about 100 times the time of an assignment statement or an integer add. On the WARP/WIMP combination a floating point multiply is about the same speed as an integer addition. In the WARP's processors it takes 4 times as long to do an integer add as a floating point multiply since one must float the integers add the results and fix the answer. The WIMP can do the addition at the same speed as a floating point operation on the WARP (modulo the parallelism).

Of the 128 available processors on the Butterfly I assume that several of them are dedicated to I/O devices or bookkeeping overhead, thus only 100 processors are available for applications code. Each processor on the Butterfly is assumed to have .5 Megabytes of memory available to it most of which can be used for data.

3. Descriptions of Edge Template Matching Algorithms

Template matching is the process of finding out where a specified structure occurs in an image. Thus template matching can be used to look for specific objects in an image. In this paper the problem of template matching is applied to the domain of edge images. For edge images simple calculations can be used to detect if a template truly matches an image at a particular point. Template matching has been applied directly to gray-level images. Here normalized correlation is the calculation that calculates the degree of match between the template and the image. Template matching on gray-level images is further described in [4].

Three algorithms will be examined for edge template matching in this paper. They are shift and add correlation, Fourier based correlation and the generalized Hough transform.

Often edges come with weights that indicate something about the strength of the signal that caused the edge. A minor change to each of the algorithms presented here results in an algorithm that takes advantage of these weights. Such changes do not effect the efficiency of these algorithms. Thus this issue is irrelevant to the paper, but is discussed further in [3].

3.1. Correlation

Correlation is historically the standard method for template matching [4]. Correlation is the sum of the pointwise products of the shifted template and the image. For binary images calculating the correlation gives the number of points where both the image and the template are one. If the template is binary and the image's elements are numbers the correlation gives the sum of the values of the image function where the corresponding template point is one.

The most straightforward algorithm to compute the correlation is to shift the template across the image and calculate the sum of products after each shift. This algorithm can be parallelized in two different ways. One method simply runs the template over different sections of the image simultaneously. The second pipelines the sum of products computation, doing several simultaneously.

The main change to the shift and add algorithm for the edge template domain is to substitute comparison for multiplication as the pointwise operation. The shifted template is compared pointwise to the image. The number of points where two edges have the same orientation is counted. The result is the degree of match between the template and the image at that point.

Another implementation of correlation uses the Fourier transform. The Fourier transform of the correlation of the image and template is the pointwise product of the Fourier transform of the image and the Fourier transform of the mirror image of the template [2]. If the transform of the tem' late is precomputed then the speed of Fourier based correlation is independent of the size of the template. A technique for using this on edge images is described later in this section.

3.2. The Hough Transformation

The Hough transformation was developed by P. Hough [10] for detecting curves in bubble chamber photographs. It was adapted to the problem of general template matching by Dana Ballard at the University of Rochester [1]. The technique that implements template matching is the generalized Hough transform or gHough for short.

Generalized Hough maps significant points of the image (many kinds of images such as edge images have sparse significant points) to template elements it can match. For each tentative match one is added to the value of the position of the template that would make the match possible. The template positions that accumulate many matches are the places where the template closely matches the image. The advantage of the generalized Hough transform is that only the significant points of the image cause matching computations and that the technique can proceed entirely by table lookup.

3.3. Edge Templates

Template matching works especially well on edge images since many of the problems that plague linear template matching on image intensities are not relevant here. One such problem is that on an intensity image a bright spot will cause correlation to have a high value near that point regardless of whether the image at that point matches the template. An image of thresholded edges has no "bright spots". This is because the comparison operation gives equal weight to all successful comparisons.

An example of an application of edge templates is circle detection. It is important for certain applications to find objects with circular boundaries in images. Figure 2 is a template for a circle 5 pixels wide.

Figure 2
Template for an edge image of a circle of radius 3

y/x	0	1	2	3	4	5	6
0	.	.	.	$\frac{3\pi}{2}$	.	.	.
1	.	$\frac{7\pi}{4}$	5.18	.	4.25	$\frac{5\pi}{4}$	.
2	5.96	5.82	.	.	.	3.61	3.46
3	0	.	.	.	.	.	π
4	.32	.46	.	.	.	2.68	2.82
5	.	$\frac{\pi}{4}$	1.11	.	2.03	$\frac{3\pi}{4}$	.
6	.	.	.	$\frac{\pi}{2}$	.	.	.

The entries that are just "." are positions that are not matched against edges. The result of matching this template against an image at position (x,y) in the image is the number of places that the template shifted to (x,y) has an edge that has an equal direction to (within a certain tolerance) an edge in the corresponding image element. The number thus computed is the number of edges in the image that support the statement "There is a circle of radius 3 at position (x+3,y+3)."

3.3.1. Shift and Add Correlation

This section describes an algorithm for template matching akin to correlation. The template is stored as a list of <x_displacement,y_displacement,angle> triplets. The algorithm is as follows.

```
for (x = 0 ; x < IMAGE_WIDTH ; x ++ ) /* move template across image */
  for ( y = 0 ; y < IMAGE_LENGTH ; y ++ ) /* move template up and down on image */
      for( index = 0 ; index < TEMPLATE_SIZE ; index ++ )
        if(an_edge_exists_at (
            image[
              x + template[index].x_displacement,
              y + template[index].y_displacement
              ]))
          if(
              template[index].angle
              is_close_to
              image[
                x + template[index].x_displacement,
                y + template[index].y_displacement
                  ].angle)
              {
              correlation_output[x][y] += 1;
              }
```

This algorithm does O(*IMAGE_SIZE*TEMPLATE_SIZE*) operations that index into the memory to check for the existence of an angle (2 integer adds) and O(*IMAGE_SIZE**number_of_template_edges) operations that compare the angles with the template angle.

3.3.2. Fast Fourier Based Correlation

Convolution and multiplication are Fourier pairs [2]. This means that one can convolve two functions by taking the inverse Fourier transform of the product of the two Fourier transforms of the functions. Correlating a template on an image is the same as convolving the image with the reflection of the template along the lower left to upper right diagonal. Thus one can implement a correlation by using a Fourier transform.

The Fourier transform is a transformation on complex functions. One could express an edge image by an array whose values were:

0 when (x,y) is not an edge
$e^{i\theta}$ when (x,y) is an edge with angle θ

The template could be expressed as:

0 when $(x\_displacement, y\_displacement)$ is not an edge
$e^{-i\theta}$ when it is an edge with angle θ

This results in a function where the real part of the correlation at (x,y) is the sum of

0 when the template element of the image element being matched is not an edge
$\cos\left[\theta_{image} - \theta_{template}\right]$ when the two match

The second function is only near 1 when the two angles are near each other and drops to 0 when they differ substantially. For such an algorithm to work only angles whose difference falls within the right half-plain of the complex plane can be compared. Otherwise numbers with negative real parts can be generated making a good match look bad. Thus a four pass algorithm is required. For each quadrant an edge image is generated using only edges with orientations in that quadrant. This edge image is correlated with a template that has only edges with orientations within that quadrant. Such a technique should render a good approximation to the correlation described above.

Assuming the Fourier transform of the template's edges is precomputed, the algorithm consists of

(1) generating $\cos(\theta)$ and $\sin(\theta)$ for each edge in the image.

(2) applying the Fourier transform to the 4 partial edge images.

(3) multiplying every element of the transformed images by the transformed templates.

(4) applying the inverse Fourier transform to the results.

The cost of the first part is the time for 2 square roots, 4 multiplies 2 adds, and 2 divides to the image multiplied by the number of edges in the image. The cost of the second part is 4 times the time for a Fourier transform. A Fast Fourier Transform requires two complex multiplies, 1 complex add, and 1 complex subtract (= 4 multiplies 6 adds 2 subtracts) * $\log(\sqrt{IMAGE\_SIZE})*IMAGE\_SIZE$. The cost of the third part is equal to $4IMAGE\_SIZE$ complex multiplies. The cost of the fourth part is equal to that of the second.

The finite field fast Fourier transform is a transformation for which convolution and multiplication are paired. Such a transformation can be performed using only integer arithmetic. Thus the finite field Fourier transform runs two orders of magnitude faster than the fast Fourier transform on machines without floating point hardware. However it is not clear how one uses convolution on a finite field for the purposes of edge template matching. The trick I used here took advantage of some ordering properties that do not apply to finite fields. Thus the finite field Fourier transform will not be discussed further.

3.3.3. The Generalized Hough Transformation

The generalized Hough transformation was designed with edge template matching in mind [1]. The data structure for the image that is most efficient is triplets of $(x\_position, y\_position, \theta)$. The template is stored in a table of parameters to be voted for indexed by ranges of angles Here, each table entry is a list of displacements to vote for. Figure 4 is the circle template expressed in this form (see Figure 2 for original template). The algorithm that implements the Hough transformation on these data structures is as follows.

```
for(edge = 0; edge < NUMBER_OF_EDGES_IN_IMAGE; edge++)
  {
  range = range edge_image[edge].angle falls into;
  on all ( displacement in template[range] )
       output[
          text[edge].x + displacement.x,
          text[edge].y + displacement.y
          ] += 1;
  }
```

The speed of this algorithm is about: *NUMBER_OF_EDGES_IN_IMAGE* multiplied by the average number of displacements per category multiplied by the cost of 3 integer adds.

4. Implementing Template Matching on the WARP and Butterfly

This section outlines various implementations of the three techniques outlined previously and investigate their timings.

This section assumes that the image is 1K by 1K and that the problem is to match edge direction templates. The size of the templates and the sparsity of edges in the image and templates are independent variables in the calculation. (It is assumed that the image and template has about the same percentage of edge points.) Another independent variable is the discretization of the angle space, which is the number of

Figure 4
Circle Template Stored in Generalized Hough transformation Form:

Angle Range	Displacements
$\frac{23\pi}{12}$ - 2π, 0 - $\frac{\pi}{12}$	[0,3]
$\frac{\pi}{12}$ - $\frac{\pi}{4}$	[0,4],[1,4]
$\frac{\pi}{4}$ - $\frac{5\pi}{12}$	[1,5],[2,5]
$\frac{5\pi}{12}$ - $\frac{7\pi}{12}$	[3,6]
$\frac{7\pi}{12}$ - $\frac{3\pi}{4}$	[4,5]
$\frac{3\pi}{4}$ - $\frac{11\pi}{12}$	[5,5],[5,4],[6,4]
$\frac{11\pi}{12}$ - $\frac{13\pi}{12}$	[6,3]
$\frac{13\pi}{12}$ - $\frac{5\pi}{4}$	[5,2],[6,2]
$\frac{5\pi}{4}$ - $\frac{17\pi}{12}$	[5,1],[4,1]
$\frac{17\pi}{12}$ - $\frac{19\pi}{12}$	[3,0]
$\frac{19\pi}{12}$ - $\frac{7\pi}{4}$	[2,1]
$\frac{7\pi}{4}$ - $\frac{23\pi}{12}$	[0,2],[1,2],[1,1]

different classes of angles. The discretization of the angle space affects the speed of the generalized Hough transformation technique. The edges are assumed to be evenly distributed through the area of the template.

The following notation describes some image statistics:

1M	Image size (1Kx1K)
T_E	Number of edges in template
T_W	Template width - the extent in the x dimension
T_l	Template length - the extent in the y dimension
E	Fraction of edges
A	Number of angle classes

4.1. Implementation on WARP/WIMP

On the WARP techniques have been developed for efficiently implementing convolution and Fourier transforms.

4.1.1. Implementation of Shift and Add Correlation Technique

If the template has less than 10 edges in it there is a simple systolic algorithm for it on the WARP. The equivalent algorithm for convolution is described in [8] and [11].

4.1.1.1. Implementation on Small Templates (size <= 10)

In this algorithm each processor has a template element. The input and output can be thought of as streams passing in parallel through the 10 processors. The input stream is initialized to the image edges and the output stream is initialized to 0. When a processor inputs an image element equal to its template element it adds 1 to the output stream. Shift registers skews the output stream in time so that the proper member of the input stream meets the proper member of the output stream in the right processor.

As an example suppose a template has elements at displacements 7 and 10 and nothing in between. Assume processor 1 has the element at displacement 7 and processor 2 has the element at displacement 10. There should be a shift register of size 7 before processor 1 and a shift register of size 3 between processor 1 and 2. Thus output element 0 meets input 7 at processor 1 and input 10 at processor 2. Output element 1 meets input 8 at processor 1 and input 11 at processor 2.

In this algorithm the angles are assigned to some finite set of classes represented by images and some other number indicates the nonexistence of an edge. The angle comparison and the test for the existence of an edge can be done in a single alu operation (absolute difference). Another alu operation is required to increment the output of the correlation. Even if no incrementing is necessary the time for the incrementing must be allocated so that the data rate can be determined in advance. The WIMP can not figure out whether to pause for an incrementation amid data without doing the correlation itself.

Thus for each datum time for two alu operations must be allocated on each processor. All the other operations required can be done in parallel with these operations. Thus this algorithm can consume a datum every 400ns. Thus the speed of this algorithm is .4s

4.1.1.2. Implementation on Large Templates (size > 10)

For template sizes greater than 10 a more complex algorithm must be attempted. One way to implement such an algorithm was to have each processor emulate $\left\lceil \frac{T_E}{10} \right\rceil$ processors. These issues must be addressed by this algorithm:

(1) Can the WARP's processors emulate $\left\lceil \frac{T_E}{10} \right\rceil$ processors?

(2) How fast would this algorithm run given that it can be implemented?

The most scarce resource on the WARP processor is memory. Simulating $\left\lceil \frac{T_E}{10} \right\rceil$ processors requires simulating the shift registers between them. If the edges are evenly distributed about the template the memory needed for such shift registers is $\left\lceil \frac{T_E}{10} \right\rceil$1K. This data will fit into the local memories of the WARP processors for templates of length < 20.

In this algorithm each processor accepts 2 inputs once every $\frac{2}{\left\lceil \frac{T_E}{10} \right\rceil}$ cycles and outputs at an equal rate. Thus its execution time is $1\text{M}\left\lceil \frac{T_E}{10} \right\rceil 400$ ns.

4.1.1.3. Algorithm for Template Lengths > 20

The WIMP's memory allows an algorithm for the WARP/WIMP combination that will work for template length's > 20. This algorithm has the WARP run the first 10 of the T processors required by the simple algorithm. The output of such a run and a shifted image is sent back to the WARP, which is simulating the next 10 processors using the output of the last 10 processors as input. The algorithm iterates until the T processors have been simulated. The final output is the output that would have been created by a T processor WARP executing the algorithm for small templates.

Such an algorithm can run as fast as the small template convolution algorithm for each stage, because it requires that two inputs be sent to the WARP every 400ns which is within the capacity of the WIMP. The speed of this algorithm is $1\text{M}\left\lceil \frac{T_E}{10} \right\rceil 400$ ns. This is as fast as the previous routine. Thus this is probably the algorithm of choice for all templates (since the algorithm for small templates is just a special case).

4.1.2. Implementation of Fourier Based Correlation

The 1K Fast Fourier Transform can be implemented in a pipeline on the WARP in a way that takes full advantage of the WARP's parallelism using techniques described in [8]. To use the Fourier transform for correlation one need first transform each of the rows of the original image and then transform each of the columns of the output. Thus 2K Fourier transformations are needed to complete the operation. 2K inverse transformations need to be run to reverse the transformation. This algorithm uses the WARP as a 10 stage pipeline. Each processor is devoted to a particular iteration of the fast Fourier transform algorithm. This pipeline must be flushed after the rows are transformed and after the columns are transformed. Thus the time to do the transformation to and from is 4K + 20 times the cost of a single iteration of the algorithm. Each of these transforms are done to an image consisting of edges whose angles that fall respectively into the 4 quadrants. Thus a total of 16K + 20 column transformations are necessary.

The fast Fourier transform is implemented by an algorithm that does $\log(n)$ iterations for n points of data. The parallelism of the WARP can be used for such an algorithm by having the $\log(n)$ iterations proceed in parallel on the different processors. Within each of the $\log(n)$ iterations the results are taken in pairs. Each pair of numbers generates a new pair of numbers that are inserted elsewhere in the results. To generate a new pair of complex numbers from an old pair one complex multiply, one complex add and one complex subtract are required. Therefore the cost of one stage of a Fourier transform on 1K points is 512 complex multiplies, adds and subtracts. A complex multiply requires 4 multiplies and 2 alu operations. A complex add or subtract requires 2 alu operations. The multiplies and alu operations can be done in parallel. Thus the entire procedure requires 1200ns per pair. This is the determining cost of the iteration since all other operations can proceed in parallel. Thus the speed of one iteration of the algorithm is 600K ns. This is the speed of the algorithm when the iterations are proceeding in parallel on different machines. Thus the speed of a Fourier transform of a 1K by 1K image is 2406M ns. The speed of the inverse is similar.

The main constraint on the speed of the multiplication of the transformed template and the image is the I/O bandwidth for moving the complex numbers to and from the WARP or WIMP's multipliers and adders. Two complex numbers take 800ns to be retrieved from memory. This is done to the 4 transformed images. Running the inverse transform on the 4 product images and summing the real parts of the results results in the correlation of the two images. The sum process is also bound by the memory speed. It can be done as fast as the 4M of correlation results can be retrieved. Thus 800M ns are required for the summing process and 3200M ns are required for the multiplication. Thus the total speed for generating the correlation of two images using the Fourier transform is 8812M ns or 8.812 s on 1K by 1K images.

4.1.3. Implementation of Generalized Hough Transformation

The generalized Hough algorithm requires that a section of memory of size $T_L T_W$ be available for random access. The WARP is not good at accessing large sections of memory randomly. Thus, the WIMP was described (though no such machine has been designed or built). The case is further complicated because the window of memory that needs to be accessed changes for each data element. Thus the WARP gives little leverage for an implementation of generalized Hough transformation. This section assumes that generalized Hough transformation is implemented entirely by the WIMP.

This section assumes that the edges have already been separated according to angle so there is no need to check the angles of the edges. In the WIMP's memory one need set up a $1K+2T_L$ by $1K+2T_W$ array. This array is where the results are accumulated. The elements of this array should be initialized to 0. Initialization takes 200 times $1K+2T_L$ times $1K+2T_W$ ns.

For each angle, iterate through the edges in the image that are of that angle. For each edge, iterate through the elements of the template that are of that angle. For each edge at (x,y) and each displacement of the template of (dx,dy), increment the output element $(x+dx,y+dy)$. The WIMP needs to read x,y,dx,dy. After that the WIMP does two adds. It then needs to do an indirect memory reference an increment and another indirect memory reference. To check to see whether the end of the list has been reached the WIMP also does one reference to registers, one decrement and check for 0. It does a check each time it generates an edge and each time it generates a displacement.

Because of the internal parallelism of the WIMP it can overlap the alu, memory and register calls. Thus the speed at which it can do one iteration of generalized Hough transformation is the speed of 6 memory accesses. Thus the speed of generalized Hough transformation on the WARP/WIMP is 1200 ns times 1M times E (since only the edges are looked at) times T_E divided by A (if the edges are equally distributed over angle space in the image) or more concisely $\frac{T_E E}{A} 1.2$ s.

4.1.4. Timing Considerations

These are the techniques described above and their speeds.

Technique for Template Matching	Speed in Seconds
Shift and Add Correlation ($T_E \leq 10$)	.4
Shift and Add Correlation ($T_E > 10$)	$.04 T_E$
Fourier Based Correlation	8.812
Generalized Hough Transform	$1.2 \frac{T_E E}{A}$

I will assume that there are 12 ranges of angles distinguished from one another between 0 and 2π. Generalized Hough is more efficient than shift and add correlation when $E < .4$. If $T_E \geq 20$ then generalized Hough will be more efficient when $E < .6 - 2/T$.

Shift and Add correlation will be superior to the Fourier transform based technique for $T_E < 220.3$.

The generalized Hough transformation will be faster than the Fourier transformation when $T_E E < 88.12$. A chart showing these effects is shown in figure 5.

4.2. Implementation on Butterfly

The parallelism of the Butterfly is arranged differently than the WARP's. Thus the techniques for using the parallelism are different. A major difference between the two machines is that memory is scarce on the WARP's processors but plentiful on the Butterfly's. Thus algorithms can be run entirely using internal memory on the Butterfly that would require access to external memory (such as the WIMP's) by the WARP [9].

Figure 5
Optimal Technique for Template Matching on the WARP
G indicates Generalized Hough transformation
C indicates Shift and Add Correlation
F indicates Fourier Based Correlation

T_E/E	.1	.2	.3	.4	.5	.6	.7	.8	.9	1
100	G	G	G	C	C	C	C	C	C	C
200	G	G	G	C	C	C	C	C	C	C
300	G	G	G	F	F	F	F	F	F	F
400	G	G	G	F	F	F	F	F	F	F
500	G	G	F	F	F	F	F	F	F	F
600	G	G	F	F	F	F	F	F	F	F
700	G	G	F	F	F	F	F	F	F	F
800	G	G	F	F	F	F	F	F	F	F
900	G	F	F	F	F	F	F	F	F	F
1000	G	F	F	F	F	F	F	F	F	F

4.2.1. Implementation of Shift and Add Correlation Technique

There are two techniques that can be used to parallelize the implementation of correlation on the Butterfly:

(1) Different sections of the image can be given to different processors that then generate the resulting correlation.

(2) The effect of different elements of the template can be calculated by different processors.

The second technique was the primary one used by the WARP. This was because the WARP was built primarily to take advantage of such pipelining. Pipelining and limits on the local memory of the WARP's processors makes programming the first technique somewhat difficult. However these are not concerns for the Butterfly. These two techniques need not be mutually exclusive either. This section examines the extent to which these techniques can be used and mixed.

To store a 1K by 1K image on the Butterfly requires that it be partitioned among many processors because no single processor has the memory to store anything larger than a sparse edge image of that size. Thus partitioning an image for processing will be explored in this section.

In [13] it was found that the partition shape that minimized communication needs and overhead between processors was the square. However once an image is partitioned no communication between processors is necessary. Thus our problem is slightly different, since there is no advantage in time between different shapes. It is assumed that the space is partitioned into a set of overlapping rectangles as in figure 6. If the length is partioned M_L ways and the width is partitioned M_W ways the space used on each processor is:

$$\left(\frac{1\mathrm{K}}{M_L}+T_L\right)\left(\frac{1\mathrm{K}}{M_W}+T_W\right)$$

Thus the total space used is $1\mathrm{M}+1\mathrm{K}T_L M_W+1\mathrm{K}T_W M_L+T_L T_W M_L M_W$. For a given number of partitions this number is minimized when $M_L T_W = T_L M_W$. Thus the most efficient partitioning partitions the image into a set of rectangles proportional to the template's shape. There being no obvious benefit from any other partitioning, assume that the image is partitioned into rectangles and the correlation routine run on each rectangle.

Clearly by partitioning the image into N sections and running the algorithm over the N sections the speedup is N. The only loss occurs in the distribution of the image to the different processors. The

Figure 6
5 by 7 Rectangles Overlapped for a 2 by 4 Operator
Each Rectangle is Outlined by a Character

```
%     %     %     %*    %*    *     *+    *+    +     +     +
%                 *     %           +     *                 +
%                 *     %           +     *                 +
%/    /     /     */~   %/~   ~     +~&   *~&   &     &     +&
%/                *~    %/          +&    *~                +&
%/                *~    %/          +&    *~                +&
%/    %     %     %*~   %*/   *     *+&   *+~   +     +     +&
/                 ~     /           &     ~                 &
/                 ~     /           &     ~                 &
/     /     /     /~    /~    ~     ~&    ~&    &     &     &
```

slowdown that occurs because of the distribution has to be smaller than that resulting from sending every element of the image through several processors as one would have to do if pipelining was used.

Checking for equality requires one subtraction and one test and if the test is true then an extra addition is required. The times for this calculation is 5000ns when the test is false and an extra 4300ns when the test is true. The time needed for doing convolution is:

$$0.05T_E+.043T_E\,E+9+.08T_L\left(\frac{T_L}{T_W}\right)^{\frac{1}{2}}+.08T_W\left(\frac{T_W}{T_L}\right)^{\frac{1}{2}}+.008T_L\,T_W\mathrm{s}$$

When the template size is large the portion of the image stored in each processor may exceed the amount of available memory on the processor. Only a byte is needed to store an angle. 500 K bytes are available on each processor for data storage. In the general case a 32 bit integer is required to store each output. These are the memory requirements:

(1) 40K bytes for output

(2) 9T bytes for template

(3) $\left(\frac{1K}{M_L}+T_L\right)\left(\frac{1K}{M_W}+T_W\right)$ for input.

Thus

$$9T+1K\frac{T_L}{100}\left(\frac{T_W}{T_L}\right)^{\frac{1}{2}}+1K\frac{T_W}{100}\left(\frac{T_L}{T_W}\right)^{\frac{1}{2}}+T_L\,T_W<450K$$

If one assumes that $T_L=T_W$ and that $T_E=E\,T_L\,T_W$ then this inequality reduces to :

$$\left(9E+1\right)T_L^2+20T_L<450K$$

Thus for all $T_L < 212$ this technique fits into the processors' memory. If E is smaller than 1 then proportionately larger templates can be used. Thus for all reasonably sized templates such a technique can be used.

4.2.2. Implementation of Fast Fourier Correlation

Fast Fourier correlation requires that 4K Fourier transforms be done on the columns of the image and 4K Fourier transforms be done on the rows of the result. Doing 40 columns in each processor will not strain the memory of any processor. The fast fourier based algorithm requires little communication since the 1K by 1K of data need only be transmitted three times. The 1K by 1K of transformed template can be prestored into the processors if many images need to be correlated.

Assume that the image starts out as an array of complex numbers. Each complex number is two 32 bit floating point numbers. The communication cost is less than the cost of transmitting 1M of 64 bit information 3 times since the second and third times the transmission may occur in parallel. Assuming a scheme can be developed to minimize contention the Butterfly's processors should be able to send each complex number in 20000ns. Thus the communication time is 10.400s. This time is dominated by the original cost of getting the 1M of 128 bit words out to the processors in the first place.

Doing a Fourier transform or an inverse transform to a column or row requires 10 iterations. In each iteration the numbers are taken in pairs and on each pair one complex multiply and one complex add and one complex subtract is required. A complex multiply requires 4 multiplies and two adds. Each complex add or subtract requires 2 adds (for the purposes of this paper a floating point subtract is considered to have the same cost as an add). Thus each iteration require $512\left[4time\_of\_multiply+6time\_of\_add\right]$. Each processor will do 10 iterations on 40 rows or 40 columns 4 times. Thus the speed of the computation will be $819200\left[4time\_of\_multiply+6time\_of\_add\right]$. The time of a multiply is 475 us and the time of an add is 295 us. Thus the computation speed is 3006.464 s. Thus the time required to run a Fourier transform based correlation on a 1K by 1K image is 3006.464 s.

4.2.3. Implementation of Generalized Hough Transformation

Partitioning the data for the generalized Hough transform in a naive way does not work because it requires each processor to store the entire output array. There is not enough memory on each processor to store such an array.

If T_L and T_W are small enough then the input and output image can be partitioned as in the simple implementation of correlation. The output is a set of overlapping rectangles that generate the final output array when added together. A simpler technique is to vote for some points using remote memory references. Using remote memory references has problems associated with synchronization, since no two processors can be allowed to increment the same point simultaneously. Thus the only way to implement generalized Hough transformation is to partition the output array among the processors.

Assume that the output array is partitioned into $100\left[\frac{T_L}{T_W}\right]^{\frac{1}{2}}$ by $100\left[\frac{T_W}{T_L}\right]^{\frac{1}{2}}$ rectangles. If the output and input data can be stored on each processor the resulting time of computation for generalized Hough transformation is:

$$0.05E\frac{T_E}{A}+8E\left[\left(\frac{T_W}{T_l}\right)^{\frac{1}{2}}+.01T_W\right]\left[\left(\frac{T_L}{T_W}\right)^{\frac{1}{2}}+.01T_L\right]$$

If $T_L=T_W$ and that $T_E=T_L\ T_W\ E$ if you are storing M by M of the image in each processor then the space used each processor is :

(1) $5\,T_L^2\,E$ bytes for the template.

(2) 40000 bytes for output.

(3) $5\frac{M^2E}{A}$ bytes for the input.

This means that

$$92\mathrm{K}\,\frac{A}{E}-T_l^2A>M^2$$

Thus:

$$92\mathrm{K}\,\frac{A}{E}>T_l^2A+\left[T_l+100\right]^2$$

With $A = 12$ as before, then

$$1.094M>13T_l^2+200T_l$$

If T_l < 282 it can be guaranteed that there will be enough memory. If E is smaller than 1 then proportionately larger templates can be handled. Thus for template sizes used in most early vision applications such a technique fits into memory.

4.2.4. Timing Considerations

These are the techniques described above and their times. Assume that $T_L = T_W = \left[\frac{T_E}{E}\right]^{\frac{1}{2}}$ and $A = 12$.

Technique for Template Matching	Speed in Seconds
Shift and Add Correlation	$.05T_E + .043T_E\,E + 8 + .16\left[\frac{T_E}{E}\right]^{\frac{1}{2}} + .008\frac{T_E}{E}$
Fourier Based Correlation	3006.464
Generalized Hough Transform	$.004T_E\,E + 8E + .16\left[T_E E\right]^{\frac{1}{2}} + .0008T_E$

The generalized Hough transformation is always more efficient than shift and add correlation, since it does no comparisons and uses more efficient data structures. The equations above show that the generalized Hough transformation is more efficient than the fast Fourier transform when

$$T_E > \frac{2\left[3006-8E\right]\left[.004E+.0008\right]+.0256E+\left[\left[2\left[3006-8E\right]\left[.004E+.0008\right]+.0256E\right]^2-4\left[3006-8E\right]^2\left[.004E+.0008\right]^2\right]^{\frac{1}{2}}}{2\left[.004E+.0008\right]^2}$$

Figure 7 charts this result. Note that generalized Hough transformation is the most efficient algorithm for T_E up to 4000 while on the WARP the Fourier based technique is the most efficient for T_E over 200.

5. Changing the Speed of Instructions

The results so far suggest that the relative timings of the different instructions in the processors has the most important effect on the choice of algorithm given an image and template. This section determines how the choice of algorithms is modified if the WARP or the Butterfly are modified to change their instruction timings. To simplify (and thus better understand) the results of such a comparison assume that there are two classes of operations occurring in the processors. Every operation in a class takes the same amount of time.

Class C1

memory reference, remote memory reference, integer subtraction, integer addition

Figure 7
Optimal Technique for Template Matching on the Butterfly
G indicates Generalized Hough transformation
F indicates Fourier Based Correlation

T_E/E	.1	.2	.3	.4	.5	.6	.7	.8	.9	1
400000	G	G	G	G	G	G	G	G	G	G
800000	G	G	G	G	G	G	G	F	F	F
1200000	G	G	G	G	F	F	F	F	F	F
1600000	G	G	F	F	F	F	F	F	F	F
2000000	G	F	F	F	F	F	F	F	F	F
2400000	G	F	F	F	F	F	F	F	F	F
2800000	F	F	F	F	F	F	F	F	F	F
3200000	F	F	F	F	F	F	F	F	F	F
3600000	F	F	F	F	F	F	F	F	F	F
4000000	F	F	F	F	F	F	F	F	F	F

Class C2

floating point multiply, floating point add, floating point subtract

Note that that the WARP processors have only floating point operations and thus must do angle comparisons with floating point subtracts. The WIMP and the Butterfly can use the C1 integer subtraction though. In this section a C1 instruction takes $T1$ time and a C2 instruction takes $T2$ time. Assume that $T1$ is always smaller than $T2$.

5.1. Pattern Matching on the Generalized WARP Architecture

This section investigates the speed of various algorithms on the generalized WARP. A generalized WARP is the same as the WARP described above except for the new assumptions about the speed of different operations.

The limited memory available on the WARP's processors constrained the implementation of template matching most severely. The interprocessor bandwidth also constrained the choice of operation in a more subtle manner. These are left unchanged in the generalized WARP. Thus the same algorithms should be used as before. The only difference is that the WIMP in this model can do its arithmetic at a different speed than the WARP. Thus the choice of implementation for algorithms is constrained in the same manner.

5.1.1. Implementation of Shift and Add Correlation

5.1.1.1. Small Templates (size <= 10)

In the small template algorithm each element of the image is compared to a stored angle using a floating point subtract. Then if the two angles are equal, the window output is incremented. Necessary memory references are done in parallel with the arithmetic operations. The speed of this algorithm is simply $2\,T2\,M$.

5.1.1.2. Large Templates (size > 10)

The WARP algorithm runs the small convolution algorithm $\left\lceil \frac{T_E}{10} \right\rceil$ times. Thus the speed of convolution is $\left\lceil \frac{T_E}{10} \right\rceil 2\,T2\,M$.

5.1.1.3. WIMP Correlation

For every combination of template and image elements two alu operations are required for the convolution. Two counters must be maintained in the registers to keep track of the locations in the image and the template. Maintaining these counters requires another 4 alu operations. Since the alu is the scarce resource for the application of the template the time that the WIMP takes for the operation is $6\,T1\,T_E\,M$.

5.1.2. Implementation of Fast Fourier Correlation

The best algorithm for the fast Fourier correlation is the same on the WARP and the generalized WARP. One iteration of the algorithm requires 6 alu operations, 4 multiplies and 4 memory references per pair of numbers per processor. Thus the speed of one iteration (in which 1 step of 10 Fourier transforms is performed) is $3000\,T2$. Thus the time for a 2D Fourier transform on the 4 1K by 1K images is $48.06M\,T2$. The time for the inverse is the same. The product of the Fourier transform of the template and the transform of the image can be done on the WIMP. For each element of the image two memory references are necessary to access the template and the image and one multiplication. The speed of the multiplication is $\min\left[2\,T1, T2\right]$M. Thus the time for the entire algorithm is 48.06M $T2+4\min\left[2\,T1, T2\right]$M.

5.1.3. Implementation of Generalized Hough

Section 4.1.3 showed that the WARP's parallelism could not be used with the generalized Hough transform. Thus in this section the time for running the generalized Hough transformation on the WIMP is determined. All the operations involved in doing generalized Hough transformation are in class C1. As determined before the scarce resource is memory in doing the Hough transformation. Thus the speed of one iteration of generalized Hough transformation requires 6 memory references that take $6\,T1$. A generalized Hough transformation on an image requires $\frac{T_E\,E}{A}$M iterations. Thus the speed of Generalized Hough transformation is $6\,T1\frac{T_E\,E}{A}$M.

5.1.4. Timings

These are the techniques described above and their speeds:

Technique for Template Matching	Speed in Seconds
Shift and Add Correlation (on WARP)	$\left\lceil\frac{T_E}{10}\right\rceil 2\,T2$M
Shift and Add Correlation (on WIMP)	$6\,T1\,T_E$ M
Fourier Based Correlation	48.06M $T2+4\min\left[2\,T1, T2\right]$M
Generalized Hough Transform	$6\,T1\frac{T_E\,E}{A}$M

In the following sections different ratios of $T1$ and $T2$ are considered regarding which algorithm to use for template matching.

5.1.4.1. T1:T2 = 1

This ratio is similar to the WARP's. The shift and add correlation technique is best done on the WARP. The shift and add correlation is superior to the Fourier correlation for $T_E < 240.3$. With $A=12$ the generalized Hough transformation is superior to shift and add correlation when $E < 4$. The generalized Hough transformation is superior to the fast Fourier correlation when $T_E\,E < 96.12$. Figure 8 charts these effects.

5.1.4.2. T1:T2 = 4

The shift and add correlation technique is best done on the WARP. The shift and add correlation is superior to the fast Fourier correlation for $T_E < 250.3$. With $A=12$ the generalized Hough transformation is always superior to shift and add correlation in this section. The generalized Hough transformation is superior to the fast Fourier correlation when $T_E\,E < 400.16$. Figure 9 charts these effects.

5.1.4.3. T1:T2 = 16

The shift and add correlation technique is best done on the WIMP. The shift and add correlation is superior to the fast Fourier correlation for $T_E < 128.49$. With $A=12$ the generalized Hough transformation is always superior to shift and add correlation in this section. The generalized Hough transformation is superior to the fast Fourier correlation when $T_E\,E < 1541.92$. Figure 10 charts these effects.

Figure 8

Optimal Technique for Template Matching on Generalized WARP (T1:T2 = 1)

G indicates Generalized Hough transformation

C indicates Shift and Add Correlation

F indicates Fast Fourier Correlation

T_E/E	.1	.2	.3	.4	.5	.6	.7	.8	.9	1
100	G	G	G	C	C	C	C	C	C	C
200	G	G	G	C	C	C	C	C	C	C
300	G	G	G	F	F	F	F	F	F	F
400	G	G	F	F	F	F	F	F	F	F
500	G	F	F	F	F	F	F	F	F	F
600	G	F	F	F	F	F	F	F	F	F
700	G	F	F	F	F	F	F	F	F	F
800	G	F	F	F	F	F	F	F	F	F
900	G	F	F	F	F	F	F	F	F	F
1000	F	F	F	F	F	F	F	F	F	F

Figure 9

Optimal Technique for Template Matching on Generalized WARP (T1:T2 = 4)

G indicates Generalized Hough transformation

C indicates Shift and Add Correlation

F indicates Fast Fourier Correlation

T_E/E	.1	.2	.3	.4	.5	.6	.7	.8	.9	1
400	G	G	G	G	G	G	G	G	G	G
800	G	G	G	G	G	F	F	F	F	F
1200	G	G	G	F	F	F	F	F	F	F
1600	G	G	F	F	F	F	F	F	F	F
2000	G	G	F	F	F	F	F	F	F	F
2400	G	F	F	F	F	F	F	F	F	F
2800	G	F	F	F	F	F	F	F	F	F
3200	G	F	F	F	F	F	F	F	F	F
3600	G	F	F	F	F	F	F	F	F	F
4000	G	F	F	F	F	F	F	F	F	F

Figure 10
Optimal Technique for Template Matching on Generalized WARP (T1:T2=16)
G indicates Generalized Hough transformation
C indicates Shift and Add Correlation
F indicates Fast Fourier Correlation

T_E/E	.1	.2	.3	.4	.5	.6	.7	.8	.9	1
1600	G	G	G	G	G	G	G	G	G	F
3200	G	G	G	G	F	F	F	F	F	F
4800	G	G	G	F	F	F	F	F	F	F
6400	G	G	F	F	F	F	F	F	F	F
8000	G	F	F	F	F	F	F	F	F	F
9600	G	F	F	F	F	F	F	F	F	F
11200	G	F	F	F	F	F	F	F	F	F
12800	G	F	F	F	F	F	F	F	F	F
14400	G	F	F	F	F	F	F	F	F	F
16000	F	F	F	F	F	F	F	F	F	F

5.1.4.4. T1:T2 = 64

The shift and add correlation technique is best done on the WIMP. The shift and add correlation is superior to the fast Fourier correlation for T_E<512.97. With $A=12$ the generalized Hough transformation is always superior to shift and add correlation in this section. The generalized Hough transformation is superior to the fast Fourier correlation when T_E/E<6155.68. Figure 11 charts these effects.

5.2. Pattern Matching on the Generalized Butterfly Architecture

This section describes timings for pattern matching on the generalized Butterfly. The implementations on the Butterfly described previously took full advantage of the full parallelism of the Butterfly. There is no obvious way to trade C1 for C2 operations while implementing these algorithms. Thus the best implementation of the techniques on the generalized Butterfly are the same as the implementation on the real Butterfly. Only the timings will differ.

Figure 11
Optimal Technique for Template Matching on Generalized WARP (T1:T2=64)
G indicates Generalized Hough transformation
C indicates Shift and Add Correlation
F indicates Fast Fourier Correlation

T_E/E	.1	.2	.3	.4	.5	.6	.7	.8	.9	1
6400	G	G	G	G	G	G	G	G	G	F
12800	G	G	G	G	F	F	F	F	F	F
19200	G	G	G	F	F	F	F	F	F	F
25600	G	G	F	F	F	F	F	F	F	F
32000	G	F	F	F	F	F	F	F	F	F
38400	G	F	F	F	F	F	F	F	F	F
44800	G	F	F	F	F	F	F	F	F	F
51200	G	F	F	F	F	F	F	F	F	F
57600	G	F	F	F	F	F	F	F	F	F
64000	F	F	F	F	F	F	F	F	F	F

5.2.1. Implementation of Shift and Add Correlation

To implement the shift and add correlation technique, first store the overlapping rectangles on the various processors. Storing these rectangles costs:

$$100\left(100\left(\frac{T_L}{T_W}\right)^{\frac{1}{2}}+T_W\right)\left(100\left(\frac{T_W}{T_L}\right)^{\frac{1}{2}}+T_L\right)T1$$

Then the convolution must be applied to all 1M points of the image. Thus each application of convolution require T comparisons and in the $\frac{1}{A}$ of the cases increments (2 memory references and an add), and three memory references to access the template, the image and the output. Thus the convolution altogether requires:

$$100\left(100\left(\frac{T_L}{T_W}\right)^{\frac{1}{2}}+T_W\right)\left(100\left(\frac{T_W}{T_L}\right)^{\frac{1}{2}}+T_L\right)T1+.01\mathrm{M}\left(4+\frac{3}{A}\right)T1$$

5.2.2. Implementation of Fast Fourier Correlation

In this algorithm each of the 100 processors transforms 10 columns and then 10 rows of the result 4 times. The 4 results are multiplied by the transformed template at the processors. The inverse transformation follows. It is executed by an algorithm similar to that for the fast Fourier transform. Thus 4M $T1$ is required to distribute the image originally. The two data transmission sections to get rows from columns or vice versa is .0792M $T1$. There is a way to arrange contention free rings of communicating processors on the Butterfly so contention is not a significant problem. The fast Fourier computation requires that 4 multiplies and 6 adds are required for each pair. This operation is done for 10 iterations Each datum must be referenced and output too. Thus all the fast Fourier transformations and inverse transformations will require 2M $T2$+.2M $T1$. The multiplication with the template will require three memory references (two inputs and one output) and one multiply thus require .01M $T2$+.03M $T1$. The sum of the 4 convolved images requires 3 adds per pixel. Thus doing the sum will cost .03M $T2$ Thus the total time required by the fourier based correlation is 8.07M $T2$+5M $T1$

5.2.3. Implementation of Generalized Hough Transformation

The most efficient solution to this problem is also to partition the problem. 1M $T1$ is required to distribute the image to the 100 processors. Then the algorithm is applied to the subimages. Thus the time required to do generalized Hough transformation is

$$1\mathrm{M}\,T1+10000\,E\left(1+4\frac{T_F}{A}\right)T1$$

5.2.4. Timings

These are the techniques described above and their speeds.

Technique for Template Matching	Speed in Seconds
Shift and Add Correlation	$1\mathrm{M}\left(\left(\frac{T_L}{T_W}\right)^{\frac{1}{2}}+.01T_W\right)\left(\left(\frac{T_W}{T_L}\right)^{\frac{1}{2}}+.01T_L\right)T1$

	$+.01\text{M}\left[4+\frac{3}{A}\right]T1$
Fast Fourier Correlation	$8.07\text{M}\,T2+5\text{M}\,T1$
Generalized Hough Transform	$1\text{M}\,T1+.01\text{M}\,E\left[1+4\frac{T_E}{A}\right]T1$

This table shows that the generalized Hough transformation is always faster than shift and add correlation. Thus one need only consider these two transformations when deciding which technique to implement on the generalized Butterfly. A formula that determines when the generalized Hough transformation is faster than fast Fourier correlation is:

$$E\left[1+\frac{T_E}{3}\right]T1 < 807\,T2+400\,T1$$

Figures 12,13,14,and 15 graphs this formual for different ratios of T1 and T2.

5.3. The Effect of Architecture on Choice of Algorithm

The decision formula developed for the generalized WARP and the generalized Butterfly differ significantly in two ways. The generalized WARP has a case in which shift and add correlation is preferable to the generalized Hough transformation.

On the generalized WARP the generalized Hough transformation is faster than the fast Fourier correlation technique when:

$$T_E\,E\,T1 < 96.12\,T2+16\,T1$$

On the generalized Butterfly the corresponding inequality is:

$$T_E\,E\,T1 < 2421\,T2+1200\,T1$$

Thus a generalized Butterfly acts like a generalized WARP when this relationship holds true:

Figure 12

Optimal Technique for Template Matching on Generalized Butterfly (T1:T2 = 1)

G indicates Generalized Hough transformation

C indicates Shift and Add Correlation

F indicates Fast Fourier Correlation

T_E/E	.1	.2	.3	.4	.5	.6	.7	.8	.9	1
2000	G	G	G	G	G	G	G	G	G	G
4000	G	G	G	G	G		G	G	G	F
6000	G	G	G	G	G	G	G	F	F	F
8000	G	G	G	G	G	F	F	F	F	F
10000	G	G	G	G	F	F	F	F	F	F
12000	G	G	G	G	F	F	F	F	F	F
14000	G	G	G	F	F	F	F	F	F	F
16000	G	G	G	F	F	F	F	F	F	F
18000	G	G	G	F	F	F	F	F	F	F
20000	G	G	F	F	F	F	F	F	F	F

Figure 13

Optimal Technique for Template Matching on Generalized Butterfly (T1:T2=4)

G indicates Generalized Hough transformation
C indicates Shift and Add Correlation
F indicates Fast Fourier Correlation

T_E/E	.1	.2	.3	.4	.5	.6	.7	.8	.9	1
8000	G	G	G	G	G	G	G	G	G	G
16000	G	G	G	G	G	G	F	F	F	F
24000	G	G	G	G	G	F	F	F	F	F
32000	G	G	G	G	F	F	F	F	F	F
40000	G	G	G	F	F	F	F	F	F	F
48000	G	G	G	F	F	F	F	F	F	F
56000	G	G	F	F	F	F	F	F	F	F
64000	G	G	F	F	F	F	F	F	F	F
72000	G	G	F	F	F	F	F	F	F	F
80000	G	G	F	F	F	F	F	F	F	F

Figure 14

Optimal Technique for Template Matching on Generalized Butterfly (T1:T2=16)

G indicates Generalized Hough transformation
C indicates Shift and Add Correlation
F indicates Fast Fourier Correlation

T_E/E	.1	.2	.3	.4	.5	.6	.7	.8	.9	1
32000	G	G	G	G	G	G	G	G	G	G
64000	G	G	G	G	G	G	G	F	F	F
96000	G	G	G	G	G	F	F	F	F	F
128000	G	G	G	G	F	F	F	F	F	F
160000	G	G	G	F	F	F	F	F	F	F
192000	G	G	G	F	F	F	F	F	F	F
224000	G	G	F	F	F	F	F	F	F	F
256000	G	G	F	F	F	F	F	F	F	F
288000	G	G	F	F	F	F	F	F	F	F
320000	G	G	F	F	F	F	F	F	F	F

Figure 15

Optimal Technique for Template Matching on Generalized Butterfly (T1:T2=64)

G indicates Generalized Hough transformation
C indicates Shift and Add Correlation
F indicates Fast Fourier Correlation

T_F/F	.1	.2	.3	.4	.5	.6	.7	.8	.9	1
128000	G	G	G	G	G	G	G	G	G	G
256000	G	G	G	G	G	G	G	F	F	F
384000	G	G	G	G	G	F	F	F	F	F
512000	G	G	G	G	F	F	F	F	F	F
640000	G	G	G	F	F	F	F	F	F	F
768000	G	G	G	F	F	F	F	F	F	F
896000	G	G	F	F	F	F	F	F	F	F
1024000	G	G	F	F	F	F	F	F	F	F
1152000	G	F	F	F	F	F	F	F	F	F
1280000	G	F	F	F	F	F	F	F	F	F

$$R_B = .04R_W - .49$$

Where R_B is the ratio between the time for the C1 and C2 operations on the generalized Butterfly, and R_W is the same for the generalized WARP.

The fact that the Butterfly acts like a WARP with floating point operations 25 times slower can be attributed to the inability of the generalized WARP to use most of its parallelism for the generalized Hough algorithm. The generalized WARP can do floating point multiplies and adds simultaneously. Only the fast Fourier transform correlation made use of this parallelism on the WARP.

6. Conclusion

This paper examines the effect of two parallel architectures on template matching techniques. It was found that for a specific template the algorithm works most efficiently differs greatly on the different machines. It can be deduced from these results that the choice between fast Fourier correlation and generalized Hough transformation or shift and add correlation is determined by the relative speeds of comparison operations, memory references and arithmetic operations. The actual implementation of these techniques is largely determined by the distribution of memory on the processors. The speed of communication between the processors is also critical in choosing between parallel and partitioned implementation. The structure of the interconnection network does not seem to have a direct effect on the implementation of the algorithm. Whether this is true for other problems remains to be determined.

7. Acknowledgement

It would have been impossible without the support of H. T. Kung and the WARP group at CMU and Chris Brown and Jerry Feldman and the Butterfly Group at the University of Rochester.

References

1. D. H. Ballard, Generalizing the Hough Transform to Detect Arbitrary Shapes, *Pattern Recognition 13*, 2 (1981), 111-122.

2. D. H. Ballard and C. M. Brown, in *Computer Vision*, Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1982, 24-30.

3. D. H. Ballard and C. M. Brown, in *Computer Vision*, Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1982, 125.

4. D. H. Ballard and C. M. Brown, in *Computer Vision*, Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1982, 65-70.

5. M. J. Flynn, Parallel Processor Forms of Computing Systems, in *Operating Systems an Advanced Course*, R. Bayer, R. M. Graham and G. Seegmuller (ed.), Springer-Verlag, New York, 1978, 81-97.

6. K. H.T. and M. O., Design Specifications for the CMU Warp Processor, Warp Internal Documentation , Sept., 1984.

7. K. H.T. and M. O., Warp: A Programmable Systolic Array Processor, Proc. SPIE Symp., Vol. 495, Real-time Signal Processing VII , Aug., 1984.

8. K. H.T., *Systolic Algorithms for the CMU Warp Processor*, Carnegie Mellon University, Jan., 1984.

9. J. Hong and H. T. Kung, I/O Complexity: The Red-Blue Pebble Game, 111, Department of Computer Science Carnegie-Mellon University, Febuaray 1981.

10. P. V. C. Hough, Method and means for recognizing complex patterns, U.S. Patent 3,069,654, 1962. " .nr t[3

11. W. J. and K. T., Vision on a Systolic Array Machine, Book chapter, Jan, 1985.

12. G. Kane, D. Hawkins and L. Leventhal, *6800 Assembly Language Programming*, Osborne/McGraw-Hill, Berkeley, California.

13. T. R. Kushner, A Theoretical Model of Interprocessor Communication for Parallel Image Processing, 1160, University of Maryland Computer Science Department, April 1982.

14. R. Newman-Wolfe, To Be published.

15. R. D. Rettberg, Development of a Voice Funnel System, Quaterly Technical Report No. 3, 3, Bolt Beranek and Newman Inc., August 1981.

16. D. Trissel, List of Timings for 68020 with 68881 coprocessor, Personal Communication.

8. Appendix

The figures given for the timings of the Butterfly are for a Butterfly whose nodes lack floating point hardware. Recently floating point hardware has been added to the Butterfly nodes. Thus there is a different ratio between the speed of floating point operations and the speed of fixed point. The FFT will be the faster technique for smaller examples. No nodes with floating point hardware are currently available outside BBN. I will use for the speed of floating point operations one half the speed of the 68020 with floating point hardware (since the memory has not been speeded up correspondingly) [16]. This table describes the old and new speeds:

Timings of Operations on the Butterfly

Operation (memory to memory)	Old Speed	New Speed	Speedup
integer assignment	4.30	1.6	2.7
floating point assignment	11.29	13.6	.8
integer addition	4.15	1.8	2.3
floating point addition	295.59	22.8	13.0
integer multiplication	68.24	6.6	10.3
floating point multiplication	474.76	25.2	18.8

These speeds change the ratio of $T1$ to $T2$ (as described in section ?) for the Butterfly. The enhanced Butterfly does not correspond exactly to a generalized Butterfly, since the $C1$ operations vary between 1.6 and 13.6 microseconds and the $C2$ instructions vary from 13.0 to 25.2 microseconds. Never the less the enhanced Butterfly roughly corresponds to a generalized Butterfly with a $\frac{T1}{T2}$ of 2. The original Butterfly roughly corresponded to a generalized Butterfly with a $\frac{T1}{T2}$ of 32 (except for integer multiplication which usually can be worked around).

8.1. Implementation of Shift and Add Correlation Technique

Checking for equality requires one subtraction and one test and if the test is true then an extra addition is required. The times for such a check will be 3400ns when the test is false and an extra 1800ns when the test is true. The time needed for doing convolution is:

8.2. Implementation of Fast Fourier Correlation

Fast Fourier correlation requires that 4K Fourier transforms be done on the columns of the image and 4K Fourier transforms be done on the rows of the result. Doing 40 columns in each processor will not strain the memory of any processor. Such an algorithm requires little communication since the 1K by 1K of data need only be transmitted three times. The 1K by 1K of transformed template can be prestored into the processors if many images need to be correlated.

Assume that the image starts out as an array of complex numbers. Each complex number is two 32 bit floating point numbers. The communication cost is less than the cost of transmitting 1M of 64 bit information 3 times since the second and third times the transmission may occur in parallel. Assuming a scheme can be developed to minimize contention the Butterfly's processors should be able to send each complex number in 20000ns. Thus the communication time is 10.400s. This time is dominated by the original cost of getting the 1M of 128 bit words out to the processors in the first place.

Doing a Fourier transform or an inverse transform to a column or row requires 10 iterations. In each iteration the numbers are taken in pairs and on each pair one complex multiply and one complex add and one complex subtract is required. A complex multiply requires 4 multiplies and two adds. Each complex add or subtract requires 2 adds (for the purposes of this paper a double subtract is considered to have the same cost as an add). Thus each iteration require $512\left[4\,time\_of\_multiply+6\,time\_of\_add\right]$. Each processor will do 10 iterations on 40 rows or 40 columns 4 times. Thus the speed of the computation will be $819200\left[4\,time\_of\_multiply+6\,time\_of\_add\right]$. The time of a multiply is 25.2 us and the time of an add is 6.6 us. Thus the computation speed is 125.42 s. Thus the time required to run a Fourier transform based correlation on a 1K by 1K image is 125.42 s.

8.3. Implementation of Generalized Hough Transformation

Assume that the output array is partitioned into $100\left[\frac{T_L}{T_W}\right]^{\frac{1}{2}}$ by $100\left[\frac{T_W}{T_L}\right]^{\frac{1}{2}}$ rectangles. If the output and input data can be stored on each processor the resulting time of computation for generalized Hough transfo. mation is:

8.4. Timing Considerations

These are the techniques described above and their times. Assume that $T_L=T_W=\left[\frac{T_E}{E}\right]^{\frac{1}{2}}$ and $A=12$.

Technique for Template Matching	Speed in Seconds
Shift and Add Correlation	$.034T_E+.018T_E\,E+8+.16\left[\frac{T_E}{E}\right]^{\frac{1}{2}}+.008\frac{T_E}{E}$
Fourier Based Correlation	125.42
Generalized Hough Transform	$.0017T_E\,E+8E+.16\left[T_E E\right]^{\frac{1}{2}}+.0008T_E$

The generalized Hough transformation is always more efficient than shift and add correlation, since it does no comparisons and uses more efficient data structures. The equations above show that the generalized Hough transformation is more efficient than the fast Fourier transform when

$$T_E > \frac{2\left[125.42-8E\right]\left[.0017E+.0008\right]+.0256E+\left[\left[2\left[125.42-8E\right]\left[.0017E+.0008\right]+.0256E\right]^2-4\left[125.42-8E\right]^2\left[.0017E+.0008\right]^2\right]}{2\left[.0017E+.0008\right]^2}$$

Figure 16 charts the results of this equation. Note that generalized Hough transformation is the most efficient algorithm for T_E up to 6000 while on the unmodified Butterfly the generalized Hough transformation is the most efficient for T_E under 400000 and on the WARP the Fourier based algorithm is most efficient for T_E over 200.

Figure 16

Optimal Technique for Template Matching on the Butterfly

G indicates Generalized Hough transformation

F indicates Fourier Based Correlation

T_E/E	.1	.2	.3	.4	.5	.6	.7	.8	.9	1
20000	G	G	G	G	G	G	G	G	G	G
40000	G	G	G	G	G	G	G	G	G	G
60000	G	G	G	G	G	G	G	G	G	G
80000	G	G	G	G	G	G	F	F	F	F
100000	G	G	G	G	F	F	F	F	F	F
120000	G	G	F	F	F	F	F	F	F	F
140000	G	F	F	F	F	F	F	F	F	F
160000	F	F	F	F	F	F	F	F	F	F
180000	F	F	F	F	F	F	F	F	F	F
200000	F	F	F	F	F	F	F	F	F	F

END
DTIC
9-86