On the Application of Massively Parallel SIMD Tree Machines to Certain Intermediate-Level Vision Tasks[1]

October 1985

Hussein A. H. Ibrahim
John R. Kender
David Elliot Shaw

Department of Computer Science
Columbia University
New York, NY 10027

CUCS-221-85

[1]This research was supported in part by the Defense Advanced Research Projects Agency, under contract N00039-84-C-0165, in part by the New York State Center for Advanced Technology in Computers and Information Systems at Columbia University, and in part by an IBM Faculty Development Award.

Abstract

In this paper, we examine the implementation of two middle-level image understanding tasks on fine-grained tree-structured SIMD machines, which have highly efficient VLSI implementations. We first present one such massively parallel machine called NON-VON, and summarize the cost/performance tradeoffs of such machines for vision tasks. We follow with a more detailed description of the NON-VON architecture (a prototype of which has been operational since January, 1985), and of the high-level parallel language in which our algorithms have been written and simulated. The heart of the paper consists of the description and analysis of algorithms for a representative Hough transform, and of an algorithm for the interpretation of moving light displays. Novel algorithmic techniques are motivated and described, and simulation timings are presented and discussed. We conclude that it is possible to exploit the available massive parallelism while avoiding many of the communication bottlenecks common at this level of image understanding, by carefully and inexpensively duplicating data and/or control information, and by delaying or avoiding the reporting of intermediate results.

Index terms: Vision hardware, Hough Transform, moving light displays, parallel processing

1 Introduction

An important goal for researchers in the field of image understanding is to construct computer-based vision systems that receive an image or a sequence of images from a sensory device, and output an interpretation of this input in real time. Input images with reasonable resolution contain large quantities of data, and conventional von Neumann machines may require an impractical amount of time to process this data sequentially. Image understanding applications, however, frequently involve computations that can be performed simultaneously on many or all of the image elements. Consequently, parallel machines have displayed great potential for the rapid and cost-effective execution of image understanding tasks, especially for low-level signal processing and the extraction of primitive geometric properties. However, our concern in the present paper will be with two intermediate-level tasks, where, like the high-level tasks of object recognition and scene analysis, the advantages of parallel architectures are less apparent.

A number of parallel architectures [5], [6], [13], and [15] have been proposed for application to image analysis problems. Of particular concern in this paper is the class of parallel machines characterized by a very large number of relatively small, simple processing elements (PE's), a number of which may be embodied within a single integrated circuit chip using contemporary VLSI technology. We will refer to such machines as *fine-grained* or *massively parallel*. Another dimension along which massively parallel architectures may be classified relates to the manner in which the PE's are interconnected. In this paper, we will be concerned with machines whose PE's are connected to form a *binary tree*. (Binary trees have a number of advantages from the viewpoint of VLSI implementation, including an asymptotically optimal layout, a bounded number of I/O pins per chip, and a simple and economical inter-chip interconnection scheme [9].) Finally, we will restrict our attention to machines in which all PE's simultaneously execute the same instruction on different data elements -- a mode Flynn referred to as *single instruction stream, multiple data stream* (SIMD) execution [7]. These three restrictions (fine-grained, tree-structure, SIMD) simplify machine design and construction; we show that they can also be exploited for cost-effective middle-level vision.

In order to make possible a detailed performance analysis, algorithms for a number of vision tasks were developed for a particular massively parallel machine, called NON-VON. The 63-processor NON-VON 1 prototype, which implements only some of the features of the full NON-VON architecture, was developed at Columbia University, and has been operational since January, 1985. An 8,191-processor prototype of a more recent version of the machine, called NON-VON 3, is presently under construction. While the full architecture [17] supports other interconnection topologies and execution modes, only its tree-structured communication capabilities and its SIMD mode of execution are used in the algorithms described in this paper. The current paper thus provides an evaluation of the strengths and limitations of a "pure" fine-grained SIMD tree machine, and not of the full NON-VON machine, which contains additional features that might be expected to offer significant performance enhancements in a number of vision applications.

Several parallel image understanding algorithms, spanning different levels within the process of image understanding, have been developed and tested using a functional simulator, and in some cases, a NON-VON machine instruction level simulator. In this paper, we describe SIMD tree algorithms for two commonly used and representative tasks drawn from the intermediate levels of computer vision. In particular, algorithms are presented for the Hough transform and for moving light display applications. Novel algorithmic techniques are described that effectively exploit the massive parallelism available in fine-grained SIMD tree machines while avoiding communication bottlenecks.

Both algorithms have been simulated using a functional simulator running on a VAX 11/750 augmented with a Grinnell image processor. Other image understanding tasks (not discussed in this paper) for which NON-VON algorithms have been developed and simulated include image correlation, histogramming, thresholding, connected component labeling, and the computation of the area, perimeter, center of gravity, eccentricity, and Euler number of connected components [10]. Based on simulation results, NON-VON's performance has been compared with that of other highly parallel architectures for image analysis. Certain algorithms have been shown to execute faster on NON-VON than on other

highly parallel machines having a similar cost. These performance and cost/performance advantages were seen to derive from

1. The effective use of an unusually high degree of parallelism, made possible by the machine's very fine granularity.

2. The natural mapping of hierarchical and multi-resolution techniques developed by other researchers onto NON-VON's tree structured topology.

3. The extensive use of content-addressable matching and other associative processing techniques.

4. The use of the tree to perform algebraically associative operations such as addition in time logarithmic in the number of pixels.

5. The simplicity and cost-effectiveness with which tree-structured machines can be implemented using VLSI technology.

In other cases, certain limitations of SIMD tree machines for computer vision applications were apparent. These problems fell into two categories:

1. Situations in which *multiple instruction stream, multiple data stream* (MIMD) techniques would be more effective than SIMD approaches.

2. Situations in which the root of the tree may become a significant communication bottleneck in the absence of special measures.

These advantages and disadvantages are even more apparent in the case of middle-level vision tasks considered in this paper. In the succeeding section, the NON-VON architecture is described, along with a Pascal-based parallel programming language that will be used to express the algorithms presented in this paper. In Section 3 and 4, we introduce the algorithms for implementing the Hough transform and moving light display applications, respectively. Issues related to the efficiency of both algorithms are discussed. In the concluding section, we attempt to characterize more abstractly the basis for NON-VON's speedup on these tasks. Since some of the conclusions appear to derive from considerations intrinsic to middle-level vision, our conclusions should apply to other architectures as well.

2 The NON-VON Architecture

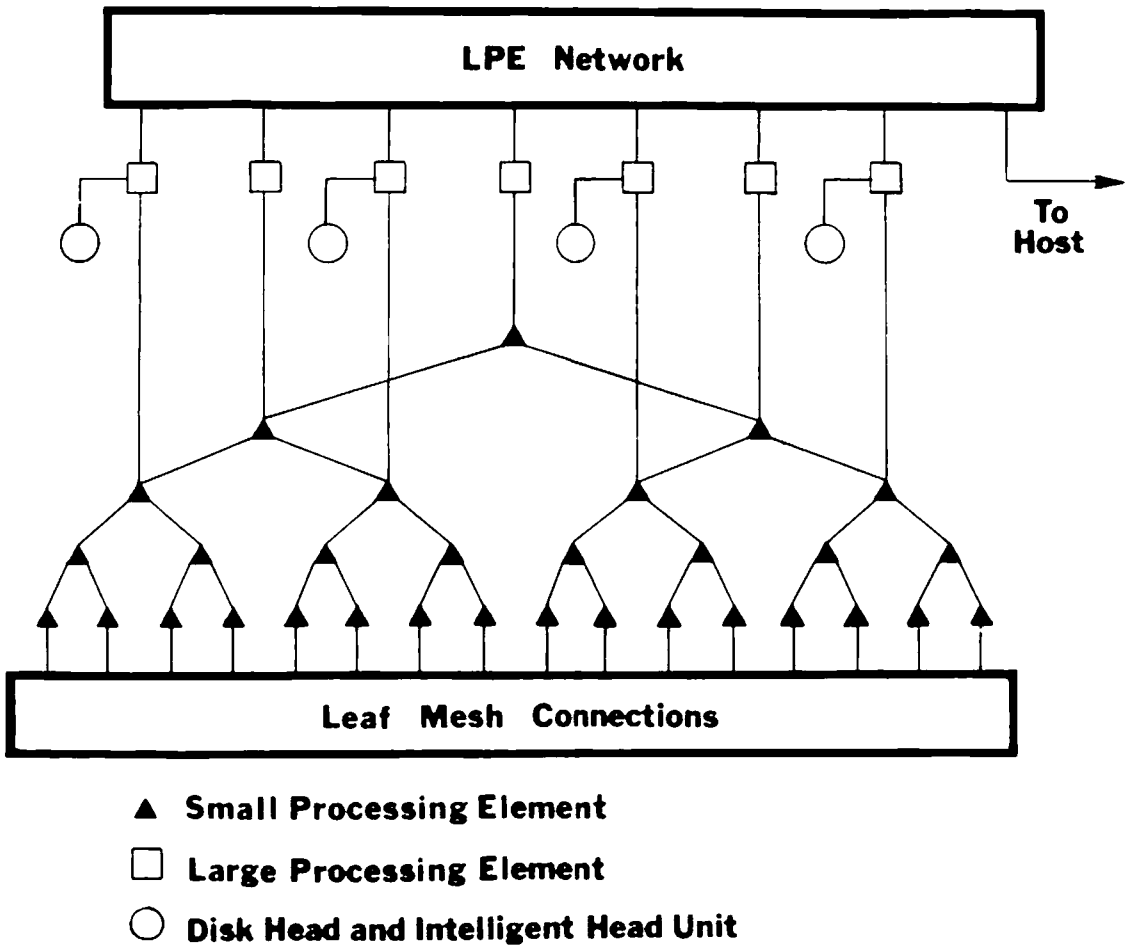
The name NON-VON is used to describe a family of massively parallel machines intended to provide high performance on a number of computational tasks, with special emphasis on artificial intelligence, database and knowledge base management, and other symbolic information processing applications. The general architecture includes a massively parallel *primary processing subsystem* (PPS) based on custom VLSI circuits, along with a *secondary processing subsystem* (SPS) based on a bank of intelligent disk drives. Figure 1 depicts the top-level organization of the general NON-VON architecture. However, only some of the subsystems depicted in this Figure are directly relevant to the concerns of this paper: in particular, we deal only with the "pure" tree subsystem.

The PPS is composed of a very large number (as many as a million, in a full-scale "supercomputer" configuration) of simple, area-efficient *small processing elements* (SPE's), which are implemented using custom VLSI circuits. Each SPE comprises a small local RAM, a modest amount of processing logic, and an *I/O switch* that permits the machine to be dynamically reconfigured to support various forms of inter-processor communication. The most recently fabricated PPS chip contains eight 8-bit processing elements. In order to maximize circuit yield, each SPE was fabricated with only 32 bytes of local RAM in the current working prototype, in a production version of the machine, however, each SPE would probably contain the maximum amount of local RAM supported by the instruction set, which is 256 bytes.

In the current version of the general NON-VON machine, the SPE's are configured as a complete binary tree whose leaves are also interconnected to form a two-dimensional orthogonal mesh. Since the present paper considers only the behavior of "pure" tree machines, however, the mesh connections will not be of concern here. In addition to tree-structured communication, the I/O switches may be dynamically configured in such a way as to support communication between any pair of nodes that would be adjacent in an inorder enumeration of the active memory tree; this is used in certain tasks requiring a linear array of processors.

The SPS is based on a bank of "intelligent" disk drives, which are connected to

Figure 1: Organization of the General NON-VON Machine



all SPE's at a particular, fixed level within the PPS tree, providing a high-bandwidth path for parallel data transfers between the PPS and the SPS. Associated with each disk head is a small amount of logic capable of dynamically examining and performing certain simple computations (hash coding, for example) on the data passing beneath it. Only the parallel I/O capabilities of the SPS/PPS interface, however, will be relevant to the concerns of this paper.

NON-VON 1 and NON-VON 3, the first two members of the NON-VON family, include a single special *control processor* (CP) at the root of the tree. The CP is capable of broadcasting instructions to be executed simultaneously by all enabled PE's. In contrast with the general architecture, NON-VON 1 and NON-VON 3 thus function for most part as SIMD machines, with all SPE's executing instructions "in lockstep". Algorithms that use this mode of execution are referred to as SIMD algorithms. NON-VON 3 is expected to execute about four million instructions per second [18]. This number is used throughout the paper to compute the time required to execute the developed algorithms.

The first member of the NON-VON family, NON-VON 1, is operational. Constructed using chips containing only a single SPE, the NON-VON 1 prototype was assembled primarily to evaluate certain electrical, timing, and layout area characteristics. NON-VON 3 [18] is based on modified chips containing multiple 8-bit SPE's. The modified chip has less area per SPE, and the instruction set has been made more powerful by generalizing register-to-register data transfers and adding more arithmetic processing power. Algorithms described in this paper are based on the NON-VON 3 architecture and instruction set.

2.1 Inter-Processor Communication

Inter-PE communication in NON-VON is supported by the I/O switch, a matrix of pass transistors that routes data between the two internal buses of the SPE and its I/O ports. The NON-VON I/O switch supports three modes of communication:

1. *Global bus communication*, supporting both broadcast by the CP to all SPE's in the PPS, as required for SIMD execution, and data transfers from a single selected SPE to the CP. No concurrency is achieved when

data is transferred from one SPE to another through the CP using the global communication instructions.

2. *Tree communication*, supporting data transfers among SPE's that are physically adjacent within the PPS tree. Instructions support data transfers to the Parent (P), Left Child (LC), and Right Child (RC) SPE's. Full concurrency is achieved in this mode, since all nodes can communicate with their physical tree neighbors in parallel.

3. *Linear communication*, in which the whole tree is reconfigured as a linear array of SPE's. This mode of communication supports data transfers to the Left Neighbor (LN) or Right Neighbor (RN) SPE's in the linear array. Linear communication is useful for applications that require a predefined total ordering of data.

Special modes of communication are employed in the execution of two NON-VON instructions. The RESOLVE instruction is used to disable all but a single SPE chosen from among a specified set of SPE's. This is an example of a hardware *multiple match resolution* scheme, in the terminology of the literature of associative processors. Upon executing a RESOLVE instruction, the CP is able to determine whether the operation resulted in any SPE being enabled. The REPORT instruction is used to transfer data from the single chosen SPE to the CP using global bus communication.

The original NON-VON architecture, which was not intended for computer vision applications, differed from other proposed highly parallel hierarchical image understanding architectures (for example, [19]) in that it did not employ any extra physical links to perform mesh neighbor communication. The "pure tree" topology of the original NON-VON machine was associated with both advantages and disadvantages. From a performance perspective, the absence of mesh connections slowed many local operations in which the output value at an image point depends on its own image value and that of neighboring points.

From the viewpoint of implementation, however, the strictly tree-structured topology had certain advantages, permitting the use of a processor embedding scheme that had a fixed chip pinout, independent of the number of embedded SPE's (unlike those involving mesh connections, in which the number of required pins grows as the square root of the number of embedded SPE's). This made it

possible to increase the size of the tree with decreasing device dimensions by simply embedding more SPE's on each chip; thus, the machine size could be increased by simply removing the old SPE chips and inserting the new ones.

Algorithms for a number of computer vision tasks, with performance comparisons assuming the presence and absence of mesh connections, are outlined in [10]. In this paper, we restrict our attention to the strictly tree-structured topology implemented in the original NON-VON 3 machine design, in the interest of clarifying the strengths and limitations of "pure" tree machines. We note that NON-VON's other special hardware features have proven useful in overcoming that residue of communication bottlenecks in "pure" tree machines that appear to resist software amelioration.

2.2 A Descriptive High-Level Language

To present the NON-VON algorithms presented in this paper, we use a PASCAL-based parallel language, referred to as N-PASCAL. It is a dialect of NV-PASCAL, which was designed for use on SIMD architectures [1]. We will now briefly describe some features of N-PASCAL that are relevant to the algorithms in this paper. One new data type and two extra constructs distinguish it from standard PASCAL. In addition, built-in functions allow the programmer to explicitly make use of the NON-VON tree communication instructions.

The new data type *vector variable* is used to express parallelism at the level of the individual data element. Vector variables refer to a set of variables, one element of which is found in each SPE, which is addressed associatively; they will be indicated by upper-case letters in the N-PASCAL procedures that follow. Standard PASCAL scalar variables reside in the CP and are sequentially addressed; they will be indicated by italics. Small bold letters will be used to refer to the reserved keywords of the language.

There are two types of statements in N-PASCAL: sequential and parallel. The sequential statements are those of standard PASCAL, while the parallel statements are those that operate on vector variables. The assignment statement therefore can be either sequential or parallel. The parallel assignment statement is executed on vector variables, concurrently in all active SPE's in the machine.

The WHERE statement is a form of parallel conditional statement that operates only on vector variables. The WHERE statement has the following syntax:

WHERE <conditional expression>
DO <statement>
[ELSEWHERE <statement>] ;

It is used to first select only those SPE's with vector variables that satisfy the boolean expression. The statement following the DO is then executed in only those SPE's. If the optional ELSEWHERE clause is included, the statement following the ELSEWHERE keyword is executed in the subset of the SPE's that failed to satisfy the original conditional expression.

Built-in functions based on the NON-VON instruction set are employed to implement operations that use the tree communication modes of the machine. Function names that start with 'N_' correspond to NON-VON machine instructions, and their parameters correspond to the arguments of these instructions. Names have been mnemonically chosen to suggest their semantics; thus, N_RESOLVE selects a single PE and N_REPORT8 is the subsequent 8-bit wide transfer of data from the selected PE to the control processor.

3 The Hough Transform

The Hough transform method is used frequently in image understanding tasks to detect the shape of object boundaries described by parametric curves. This method is based on the mathematical duality between points on a curve and the parameters of that curve. Since we will describe two differing implementations of the method, we quickly review it here in its simplest form. The methods and the analysis will be based on this simple form, also, although the properties of the implementations are easily extended to more complex versions of the transform.

In his initial work, Hough [8] described a method for detecting straight lines in an image using the slope-intercept parameterization of the line. According to this parameterization, the line equation is expressed as:

$$y = mx + c \tag{1}$$

Suppose that we have a set of image points (x_1,y_1), ... , (x_n,y_n) that have a likelihood of being on linear boundaries. In this paper, we refer to these points as *boundary points*. The Hough transform method organizes the boundary points into a set of straight lines as follows.

Consider a boundary point (x_i,y_i) in the image plane. The parameters of all lines passing through this point must satisfy the equation:

$$y_i = mx_i + c$$

This equation corresponds to a straight line in the m-c space (*the parameter space*). Thus, the set of boundary points in the image plane corresponds to a set of lines in the m-c plane. If two boundary points are on a line AB in the image plane with parameters m_1 and c_1, then the two lines corresponding to these two points in the m-c plane intersect at the point (m_1,c_1). In fact, all boundary points in the image plane on the same line AB map to lines in the m-c plane that intersect at the point (m_1,c_1). Thus, the problem of finding the set of lines in the image plane is reduced to that of finding common points of intersection of lines in the parameter space.

Much is known about this transform. A better parameterization of a straight line is suggested by Duda [4], in which the parameters θ and ρ are used, where θ is the angle of the line normal and ρ is the algebraic distance from the origin, the advantage of this parameterization is that the values of θ and ρ are bounded. The Hough transform can be extended to detect other curves of analytical parameters [11], or to detect general curve shapes using edge orientation at the image points and a reference point [2]. A memory-efficient implementation of the Hough transform on sequential machines is described in [3]. A parallel algorithm based on the Hough transform for detecting a general curve with specific orientation has been developed by Merlin et al [14]. Since we are primarily interested in exploring the efficiency of NON-VON-like machines on this method, we will not need to attempt its more refined forms.

The implementation of the Hough transform for detecting straight lines on a sequential machine involves a quantization of the parameter plane into a quadruled grid. The grid size is determined by the acceptable errors in the parameter values,

and the quantization is confined to a specific region of the parameter plane determined by the range of parameter values. A two-dimensional array (*the accumulator array*) is then used to represent the parameter plane grid, where each array entry corresponds to a grid cell. For each boundary point, the algorithm on a sequential machine increments the counts in all accumulator array entries that correspond to grid cells along the straight line in the parameter plane. After this step, grid cells corresponding to the accumulator array entries where the count exceeds a certain threshold value are selected as the set of parameters for the image straight lines being sought.

The process of incrementing accumulator array counts can be thought of as "voting" by the boundary points for the parameter values of possible curves passing through these points. The time required to execute this algorithm on a sequential machine is proportional to the number m of boundary points times the number of votes v of each point, plus the cost of scanning the grid of size s to select the maximum: $(O(s+mv))$. Memory space required is proportional to the size of the grid.

In what follows, we describe and contrast the efficiencies of two algorithms to implement the Hough transform on NON-VON. The first one is a direct parallel implementation of the standard sequential algorithm. The disadvantages of this approach are analyzed, and we describe a second approach that alleviates these problems. We assume that the boundary points have been detected by some other procedures and that the SPE's which are holding them, one boundary point per SPE, are marked using a special flag. Again, we assume the simplest case in order to highlight the machine-dependent aspects of the problem: the curves being sought are straight lines, whose equations are expressed using the slope and intercept parameters.

3.1 The Hough Transform Algorithm - A Direct Approach

First, each NON-VON SPE is associated with an accumulator grid cell in the parameter space in the following manner. The NON-VON tree SPEs are uniquely enumerated using the inorder enumeration described in [12]. The number assigned to each SPE is stored in the vector integer variable ADDR. If the parameter

space is m by c, then the address of the grid cell held by each SPE is the pair (M, C), where M is the remainder and C the integer quotient obtained when ADDR is divided by m. The N-PASCAL procedure to perform this straightforward association is described in [10], and executes in time proportional to the tree height. In effect, each grid cell is superimposed on the tree structure by decoding its position in the linear ordering of the tree nodes.

A vector integer variable COUNT is initialized to zero in all SPE's before starting the algorithm. The coordinates of boundary points in the image (still stored in the SPE's) are then reported to the CP one point at a time using the RESOLVE instruction. The reported point is then broadcast to all SPE's, which increment their vector variable COUNT by one if it satisfies the parameter space curve equation for their cell. Thus, image boundary points are retrieved sequentially although accumulator grid points are updated in parallel (albeit slowly) for each point. All boundary points have been reported when the vector variable HT is universally false. In the selection phase, each SPE whose COUNT variable exceeds a threshold value is marked, and the value of the grid cell associated with it is reported to the CP using the RESOLVE and REPORT instructions. Final selection of a maximum is made in the CP. The N-PASCAL algorithm that describes the procedure follows:

```
Procedure hough1(thr: integer);
var
  x, y, m, c: integer;
vector_var
  COUNT, X, Y: integer;
  PARM: boolean;
begin
/* 1. Initialize. */
  COUNT := 0;
  PARM := false;

/* 2. Enable all SPE's whose boundary points have not yet been reported.
Report the coordinates of a single boundary point using the RESOLVE instruction
and mark it as reported. Broadcast the point. Increment COUNT in all SPE's
in which point satisfies the equation. Repeat Step 2 until all boundary points
have been reported. N_A1 is the special flag invoked by the RESOLVE
instruction. */

  N_A1 := HT;
```

```
while N_RESOLVE(N_A1) = 1 do  begin
    where N_A1 = true do begin
        HT := false;
        N_REPORT8(XADD, x);
        N_REPORT8(YADD, y);
      end;
    X := x;
    Y := y;
    if Y = (M * X + C) then
      COUNT := COUNT + 1;
    N_A1 := HT;
  end;
```

/* 3. Mark all SPE's in which the count exceeds the threshold value and report them. */

```
where COUNT > thr do PARM := true;
N_A1 := PARM;
while N_RESOLVE(N_A1) <> 0 do begin
    where N_A1 = true do  begin
        PARM := false;
        N_REPORT8(M, m);
        N_REPORT8(C, c);
      end;
    N_A1 := PARM;
  end;

end;
```

Step 2 is executed a number of times equal to the number of boundary points b. Step 3 is executed a number of times equal to the number of curves found, which is usually less than b. Thus, the algorithm takes time proportional to the number of image boundary points ($O(b)$). The NON-VON 3 code for this procedure [10] executes approximately 500 instructions to associate parameter values with PE's, assuming that all possible values fit within the NON-VON tree. Step 2 executes about 70 instructions, of which approximately 40 instructions implement the evaluation of the straight line equation. Step 3 executes about 12 NON-VON 3 instructions for each set of parameter values found. Thus, if the image contains 1000 boundary points, the execution time of the algorithm is approximately 18 msec.

The number of SPE's required by this approach is equal to the number of grid

points. If the grid size is larger than the machine size by a factor of k, the parameter space is divided into k parts, and the above procedure is executed for each of these parts in turn. The time required to execute the algorithm in this case is $O(kb)$, independently of how the k-fold division is multiplexed into the existing time and space.

A major disadvantage of this approach is that it requires a NON-VON machine of size comparable to the grid size, despite the fact that many of the SPE's never increment their COUNT. Note also that each time a boundary point is broadcast, the curve equation has to be evaluated in each SPE. (It does, however, exploit the associative memory of the machine in searching both for boundary points and for curves above threshold.) Our second approach alleviates these problems: it uses a number of SPE's equal to the maximum number of total votes, and the curve equation is evaluated only once.

3.2 The Hough Transform Algorithm - A MSIMD Approach

Here, the NON-VON tree is configured as if it were an independent set of subtrees, with each boundary point deciding to cast its vote only in its own subtree. Votes are sequenced by broadcasting an enumeration of a cross-section of the accumulator array; in this problem, they are sequenced by broadcasting all possible quantized values of the parameter m in order. However, for each value (of m), the voting process can be performed concurrently in all the subtrees, requiring little inter-tree communication. (In analogy to uniprocessor data structures, instead of the entire tree representing the total array of grid points of size m by c, many of which are empty, each subtree now represents a distributed array of voted-for grid points of size b by m, most of which are non-empty.) Therefore, in time proportional to the quantization of a cross-section of the grid, all votes are cast and stored throughout the tree. Because of the way the votes are cast in this second approach, we refer to this algorithm as a multiple-SIMD (MSIMD) algorithm. The problem of finding the parameter values that exceed the threshold value is now equivalent to that of finding the local peaks in a distributed histogram; here the histogram is two-dimensional, m by c.

The size of these subtrees is determined by the maximum number of votes,

max_num_votes, cast by any boundary point. Boundary points are stored in the roots of these evenly distributed subtrees. This storing can be performed by several methods; the simplest (but not the most efficient) one is to report the boundary points to the CP one by one using the RESOLVE instruction, and then to broadcast them to be stored in the roots of the subtrees. This has complexity $O(b)$ as before, although the constant is much smaller. A far better way is a type of parallel "elevation" method in which image boundary points in each subtree are reported to the appropriate level, with only excess boundary points in any subtree redistributed in serial fashion over the remaining free subtrees. Multiplexing may occur, but is unlikely given that the number of boundary points tends to be a small fraction of the image.

The SPE's in these subtrees are enumerated so that each is assigned a unique address (stored in the integer vector variable ADDRESS) relative to the subtree, in the range [0, *max_num_votes*]. This enumeration procedure is similar to the address enumeration procedure described in the previous section, except that the number assigned to each SPE is the computed address modulo the subtree size; this can be done by simple shifting. Again, the time required by this procedure is proportional to the height of the subtree. In effect, the tree has become a dense two-dimensional accumulator array, addressed by boundary point number and sequential vote number.

We now describe the algorithm for storing the votes in the NON-VON tree. The vector integer variables X and Y are used to store the value of the boundary points, while the vector variables M and C are used to store the parameter values voted for by the boundary points. A scalar variable *g_m* stores the value of parameter M to be broadcast, and the scalar constant *delta_m* is the increment used to change the value of *g_m*. The scalar constant *h_subtree* is the height of the subtree. The N-PASCAL voting procedure follows:

```
Procedure hough2;
var
  i, j, g_m: integer;
vector_var
  M, C, X, Y: integer;
```

```
begin
/* 1. Initialize global variables. Enable all SPE's that are not the root of some
voting subtree.  Propagate the X and Y values of the root of the subtree
throughout the subtree. */
  i := 0;
  g_m := 0;
  where SUBTREE_ROOT = false do begin
      N_RECV8(P, XADD, X);  N_RECV8(P, YADD, Y);
      for j := 1 to h_subtree-1 do begin
          N_RECV8(P, X, X);
          N_RECV8(P, Y, Y);
        end;
    end;
/* 2. Step through the subtree addresses, storing in them the increments of
M. Now compute all corresponding C's from the curve equation. */
    while i < max_num_votes do begin
        where ADDRESS = i do M := g_m;
        i := i + 1;
        g_m := g_m + delta_m;
      end;
    C := Y - M * X;
end;
```

Step 1 is executed a number of times equal to the subtree height, log v, where v is equal to the number of votes cast by each point. Step 2 is executed exactly v times. Thus, the procedure to store the votes in the subtree takes time of $O(v)$. Note that the curve equation is evaluated only once.

In general, the evaluation of the "dependent" variables (here, c), depends on the parameter space curve, which may produce multiple values for the dependent parameters. (For example, the parameter space curve may be a circle, with two values of its counterpart of c, one each for the top and bottom arcs.) Occasionally the parameter space curve is not separable, as it is in the case of transcendental equations. It is no trouble to compute and store multiple values; in general, this adds only a constant factor. In the second case, however, it may be necessary to replace direct computation in the SPE's with the broadcast of a table of valid pairs from the CP; each SPE passively waits for a match on its first parameter, and then stores the second. This process takes time proportional to the length of the table, but it too is executed only once.

The NON-VON 3 code for the voting procedure executes approximately $8v + 25 \log v + 100$ NON-VON 3 instructions. For v equal to 32, the time required to cast the votes in the tree is thus about 0.14 msec. If the number of required votes exceeds the number of SPE's in the NON-VON tree, each SPE can be used to store more than one vote. If each SPE stores k values, the time required to execute the above procedure is $O(kv)$.

We now describe the manner in which those parameter values whose votes exceed the threshold value are found. These values occur at the local peaks of the two dimensional histogram of the votes for M and C. We assume in the following discussion that there are few such local peaks, which is usually realistic. Figure 2 shows such a histogram.

A direct approach to the identification of these local peaks would require the quantization of the two dimensional histogram space into grid cells, perhaps in a coarser manner than by the original quantization. Then, for each new grid cell, all SPE's with ordered pairs (here, of (m, c)) falling within it are marked and counted. The time required to execute this simple procedure is $O(sh)$, where s is the grid size and h is the NON-VON tree height, with the latter coming from the need to associatively add counts from subtrees into the root. Counts that exceed a threshold value are the parameter values being sought. However, a large percentage of the time in this procedure is spent counting votes in sparsely populated cells.

A more effective approach attempts to avoid such areas. It first computes a one-dimensional histogram along one parameter (c, as shown in Figure 2.) A pipelined-SIMD algorithm to compute the one-dimensional histogram is described in [10]. Only those votes are marked whose parameter is found among the small number of local peaks expected to appear in the one-dimensional histogram. A second one-dimensional histogram of the second parameter (m) is then computed for these marked votes only. Values for which there exist local peaks in the two one-dimensional histograms mark regions of activity in the two-dimensional histogram. (If the converse is expected to fail, or if no such peaks are found, the prior two-dimensional method must be used.)

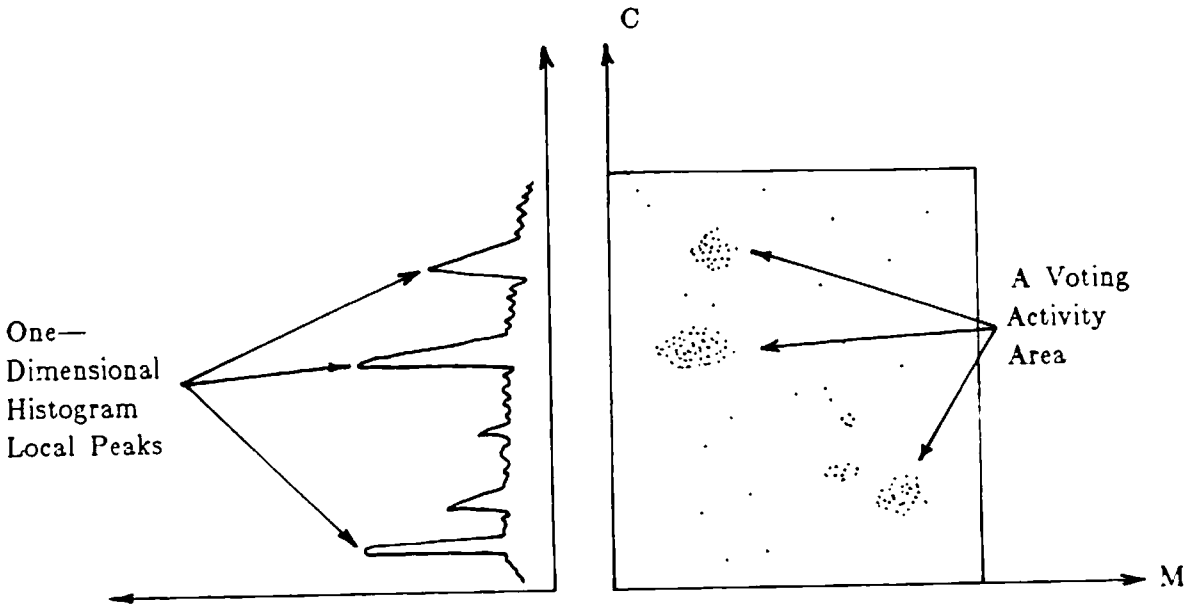


Figure 2: The Two-Dimensional Histogram of Parameter Values

This approach executes in time of $O(b_m + b_c + h)$, where b_m and b_c are the number of bins in the two one-dimensional histograms. The computation of a 32-bin one-dimensional histogram requires about 0.5 msec; peak-finding thus takes about one msec. Total execution time for the MSIMD approach is thus about 1.14 msec, which is considerably less than the time required by the direct Hough approach (18 msec for 1000 boundary points).

The algorithms described here can be extended using slight modifications to deal with parameter spaces of higher dimensions. For example, in the direct approach, if we have an n-dimensional parameter space, then each SPE will correspond to a n-dimensional grid cell in this space. In the MSIMD approach, the subtree size will correspond to that of $(n-1)$-dimensional area of the parameter space, and each SPE will store parameter values that represent cells in this sub-parameter space

3.3 Simulation Results

The two algorithms described in this section have been tested using the functional simulator. Boundary points representing straight lines in a 32 x 32 binary image, as shown in Figure 3, have been input to the simulator. The parameter space grid is a 32 x 64 grid, with m taking the values -15 to 16 and c assuming the values -10 to 53. The two-dimensional accumulator array of these lines are shown in Figure 4-a.

In the second approach, 16 votes are cast in each subtree with m varying from -7 to 8. Figure 4-b depicts the two-dimensional histogram of the votes stored in the tree. The second approach has computed the same set of straight lines found by the first approach.

3.4 Discussion

The MSIMD approach illustrates some of the advantages of duplicating data throughout the tree in order to avoid communication costs within it. No additional time cost is incurred in broadcasting a value (here, one of the quanta of m) throughout the entire tree over that of broadcasting it to one PE. The presence of many local copies allows the true parallel computation of "dependent" variables, now themselves distributed. Thus, as long as storage remains adequate

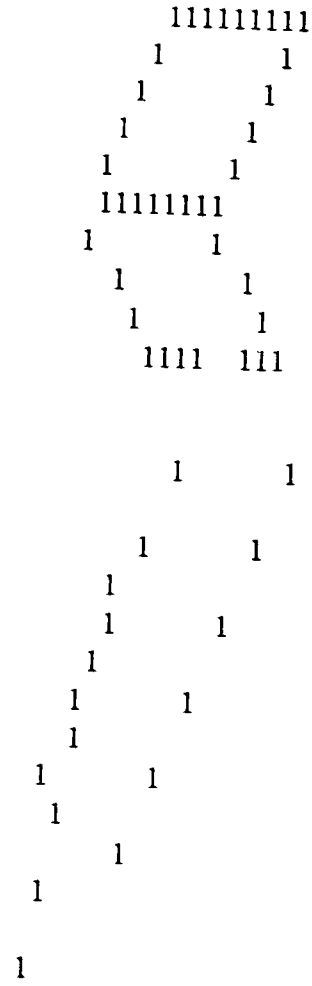
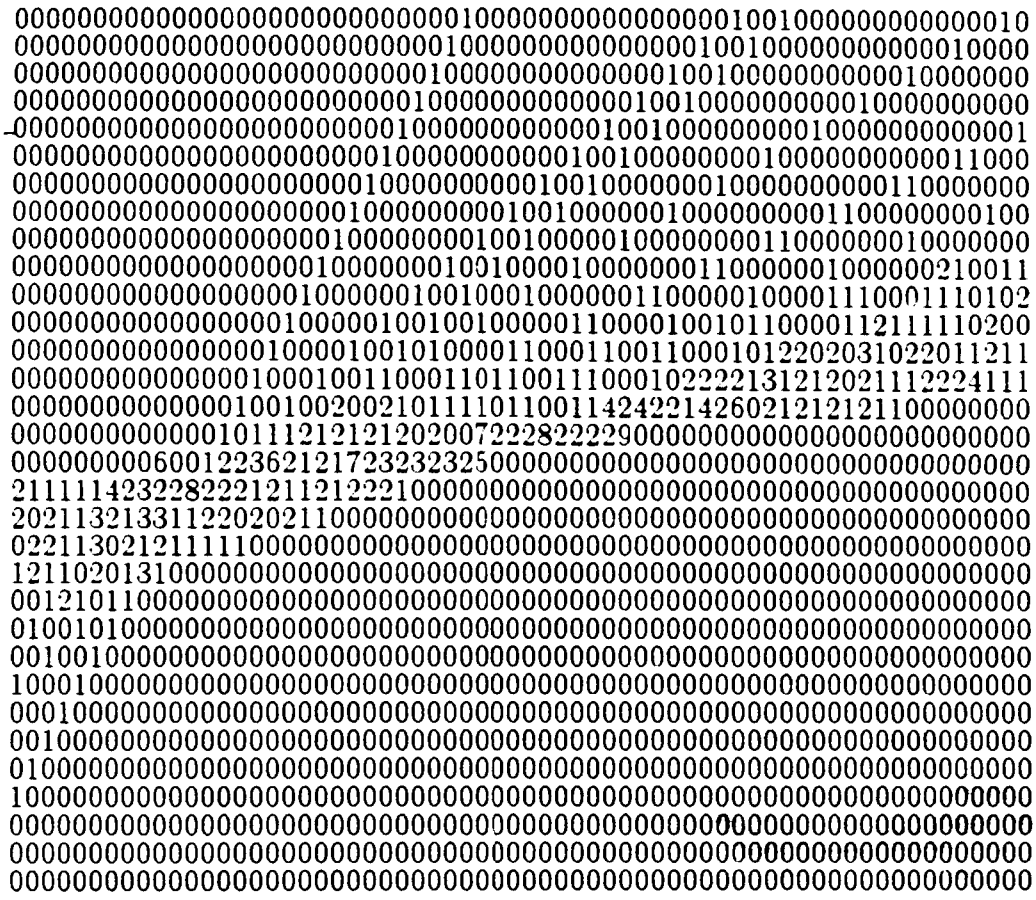
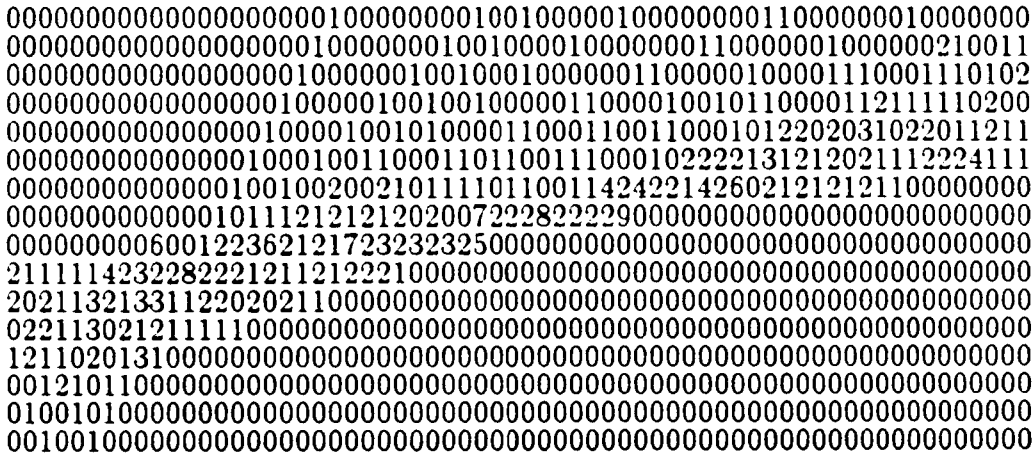


Figure 3: The Input Boundary Points



(a) The 32 X 64 Accumulator Array



(b) The 16 X 64 Two-Dimensional Histogram

Figure 4: Some Hough Transform Simulation Results

and an efficient retrieval method can be found, input redundancy can alleviate the need to precisely and uniquely distribute data (here, ordered pairs). It is this last step which becomes, in the worst case, purely sequential in a tree-based machine--as well as some other architectures).

Profligate duplication is of no help, however, if data must be precisely and uniquely retrieved (that is, sorted). It is useful only in those problems where intermediate results can be suppressed because the final result relies on an associative operation, such as addition of partial results or the calculation of a global maximum. By delaying or avoiding intermediate data transfer, tree-based communication is replaced by a final series of RESOLVE-like requests for only the most pertinent information. Most data either remains in the tree (here, ordered pairs that have sparse votes), or is only logically removed by arithmetic combination (here, the accumulated one-dimensional histogram). Again, the technique applies to other architectures as well.

It appears that more than a few middle-level vision tasks can be approached in such a duplicate-and-delay manner. A qualitatively different one follows.

4 Moving Light Displays

We now describe a NON-VON algorithm that implements the correspondence step in methods that interpret the motion of jointed objects from binary *moving light displays* (MLD). An MLD system uses only information about the position and velocity of its points for the perception of motion, and a sequence of such binary images (frames) are required for the interpretation of the object motion. The objects in these frames are typically represented by a relatively small number of points (typically less than one hundred). Rashid [16] has implemented a system, which he calls Lights, that interprets simple MLD's. The input to this system is a set of coordinate pairs corresponding to the points of the MLD.

The correspondence step is concerned with mapping the points of one frame to the next. The only information known is the position of frame points depicting parts in relative motion and the average velocity of these points based on previous frame information. A fundamental assumption is that the velocity of MLD points

varies "smoothly". This assumption can be used to predict the position of the MLD points in the next frame. The tracking algorithm computes the correspondence that minimizes the sum of differences between the expected position of each point and the actual position of the corresponding point in the next frame.

The NON-VON algorithm starts by computing an approximate solution based on the heuristic mentioned earlier. Assume that the first frame contains m points and the next frame contains n points; note that m and n may differ, since different points may be occluded in the two frames. Since we wish only to highlight the machine-dependent aspects of the computation, we assume that the number of points in both frames is equal. The initial approximation is performed by calculating for each point in the first frame that point closest to its predicted position, among those points that have not been selected yet in the second frame. This approach to compute the initial solution is basically a greedy algorithm, where the best local match among the available ones is selected. As in all greedy algorithms, the order in which points in the first frame are ordered is critical.

We would like points in the first frame that are near each other in the image to also be near to each other in the ordered set. This heuristic ordering is important to our algorithm, but it should be noted that in general a perfect linear ordering of a two-dimensional graph is not possible. There are several ways to strengthen this starting step; perhaps the easiest is by using redundant first-frame orderings. That is, if there are two points that are sufficiently near each other in the two-dimensional frame but are far from each other in the ordering, then an additional starting solution is computed by altering slightly the method for obtaining the first-frame ordering (which perhaps was itself a greedy algorithm) so that the stranded nearby neighbor is selected more immediately. This alteration can be done in many ways, including purely stochastic ones; like other heuristic algorithms its effectiveness is hard to analyze. However, there is much redundancy in the processing of any given starting correspondence in any case, as the rest of the algorithm shows.

We now apply the following procedure to each computed initial solution, which is stored in the root SPE of the tree as a m-vector of pairs of points. In order to

concentrate on the SIMD aspects of the problem, we will show its application to only one initial solution. The basic idea of the procedure is to quickly generate from the initial pairing other possible solutions to the correspondence problem, and to store these additional candidate solutions as m-vectors of point pairs in the leaf SPE's. Once in the leaves, the goodness of all correspondences can be calculated in parallel and the one with minimal distance quickly found using the tree connections in $O(h)$ time.

Additional candidates solutions are generated by permuting some of the assignments of second-frame points to other first-frame points, but only if the first-frame points involved in the permutation are near to each other in the image. Based on the ordering of first-frame points, this can be done mechanically by restricting the swapping of assignments to be among a limited contiguous subvector of a given m-vector of pairings. As an example, we briefly describe now how possible solutions containing all permutations of a frame with three elements can be computed. The initial solution in the root SPE is passed to its two children. The left child keeps the parent's solution, while the right child performs a permutation on this solution by swapping the first two assignments, as shown in Figure 5.

Again the solution is passed to the next level SPE's. The right children swap the second and third assignments in their solution. When the solution is passed to the following level, right children swap the first and third assignments. At this point solutions containing all possible permutations of the first three elements are found at the fourth level down the tree. Note that at the fourth level we have two solutions that are duplicated. This will not affect the accuracy of the procedure, but it is inefficient. In general, this process would continue until the leaf level is reached. At this point we would have $O(2^h)$ possible solutions stored in the leaf SPE's of the NON-VON tree, some of them redundant, depending on the sophistication of the control algorithm. Much work can be done on the finding of optimal strategies for the choice and order of the permutations.

We argue that the selected solution using this algorithm is a very satisfactory match. The rationale for this is two-fold. First, the initial solution is usually a

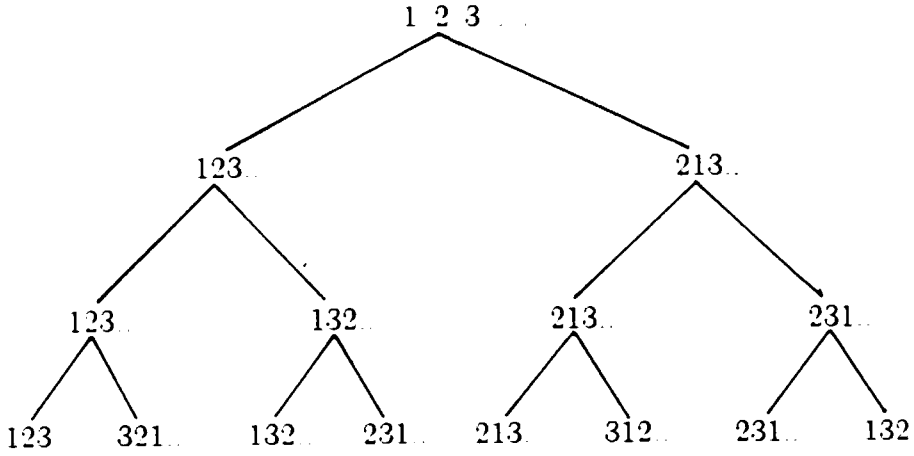


Figure 5: Permutations of the Initial Solution First Three Elements

good approximation of the desired solution, based on the heuristic assumption used in computing it. Second, as much as possible the permutations on the initial solution are performed between points that are near to each other in the image frame. That in turn insures that if there is a conflict resulting from two points in the first frame selecting the same correspondence in the second frame, then alternatives including both selections will be among the set of possible solutions.

The N-PASCAL procedure describing this algorithm for frames containing at most 16 points follows. It begins with some ancillary routines whose purpose is straightforward.

```
Procedure mld;
var
  i, j, k, delta, dist_sqr: integer;
  sol, x, y: array[1..16] of integer;
  x2, y2, xp, yp: array[1..16] of integer;
vector_var
  X1, Y1, U, V, X2, Y2: integer;
  NUM, DIST, TEMP: integer;
  XP , YP, LEVEL_NO: integer;
  S: array[1 16] of integer;
  RIGHTC, ROOT, F1, F2, N: boolean;

/* The following function finds the minimum value of a vector variable. */

function min_leaf(var MIN: integer): integer;
var
  j: integer;
vector_var
  TEMP: integer;
begin
  for j := 2 to no_levels do begin
     N_RECV8(LC, MIN, MIN);
     N_RECV8(RC, MIN, TEMP);
     if TEMP < MIN then MIN := TEMP;
   end;
  N_RECV8(LC, MIN, MIN);
  min_leaf := N_GG8;
end

Procedure swap_sol(i, j: integer);
vector_var
  TEMP: integer;
```

```
begin
  where RIGHTC = true do begin
      TEMP := S[j];
      S[j] := S[i];  S[i] := TEMP;
    end;
end;

begin

/* 1.  The first frame points (X1, Y1) and their corresponding velocity
components (U, V) are stored in leaf SPE's whose F1 is set to 1, and analogously
for the second frame points (X2, Y2).  NUM holds the ordinal position of the
point in the frame.  The time lapse between two frames is stored in the variable
delta.  Compute the predicted solution (XP, YP) using delta.  */

  XP := 0;
  YP := 0;
  mark_rc(RIGHTC);
  mark_root(ROOT);
  set_level_number(LEVEL_NO);
  where F1 = true do begin
      XP := X + U * delta;
      YP := Y + V * delta;
    end;

/* Compute the initial solution.  This is performed by reporting the point in
the first frame, and for each reported point computing the point nearest to
it in the second frame.  */

  N  := F2;
  i  := 1;
  while (i <= m) do
    begin
      where (NUM = i) and (F1 = true)
        do begin
             N_REPORT8(XP, xp[i]);
             N_REPORT8(YP, yp[i]);
           end;
      DIST := 0;
      where N = true do
        DIST := (xp[i] - X2) * (xp[i] - X2)
              + (yp[i] - Y2) * (yp[i] - Y2);
      dist_sqr := min_leaf(DIST);
      where (N = true) and (DIST = dist_sqr)
        do begin
```

```
        N := false;
        N_REPORT8(NUM, sol[i]);
      end;
    i := i + 1;
  end;
```

/* 2. Store the initial correspondence in the root SPE. */

```
  where ROOT = true do
    for j := 1 to m do S[j] := sol[j];
```

/* 3. Perform the permutations on the initial solution and store these possible solutions in the leaf SPE's. */

```
  j := 2; i := 2;
  while j <= no_levels do begin
      where LEVEL_NO = j do begin
          for k := 0 to m do
            N_RECV8(P, S[k], S[k]);
          swap_sol(i, (i-1));
        end;
      j := j + 1;
      where LEVEL_NO = j do begin
          for k := 0 to m do
            N_RECV8(P, S[k], S[k]);
          swap_sol(i, (i+1));
        end;
      j := j + 1;
      where LEVEL_NO = j do begin
          for k := 0 to m do
            N_RECV8(P, S[k], S[k]);
          swap_sol((i-1), (i+1));
        end;
      j := j + 1;  i := i + 2;
    end;
```

/* 4. Compute the sum of differences for each possible solution. */

```
  DIST := 0;
  for j := 1 to m do begin
      for k := 1 to n do
        where S[j] = k do begin
            XP := x2[k] - xp[j];
            YP := y2[k] - yp[j];
          end;
      DIST := DIST + XP*XP + YP*YP;
    end;
```

```
/* 5. Find the value of the minimum sum of differences and recover the solution
*/

  dist_sqr := min_leaf(DIST);
  where DIST = dist_sqr do
      for j := 0  to m do
        N_REPORT8(S[j], sol[j]);
end;
```

Steps 1 through 4 of the algorithm execute in time proportional to the product of the number of points in the first frame and the tree height ($O(mh)$). On the other hand, Step 5 of the algorithm takes time proportional to the product of the number of points in the two frames ($O(mn)$). Actually, if the number of frame points is comparable to the tree height, then the initial solution can be computed in the CP. This has the advantage of being able to overlap the computation of the initial solution for the following frame with the computation of the solution for the current frame. Thus, the algorithm executes in $O(\max(h,n)m)$ time.

4.1 Simulation Results

The algorithm has been functionally simulated with frames having up to six points and a tree of 10 levels, and has been found to produce optimal solutions. The first frame points have been used instead of their predicted positions. They have been ordered by starting at an arbitrary point and finding the nearest point to it. This greedy algorithm is continued until all points have been ordered.

The moving light display algorithm has been coded in NON-VON 3 machine instructions to obtain an accurate instruction count. Step 3 executes 25 instructions plus six instructions per frame point for each level in the tree, while computing the minimum solution in Step 5 executes 32 instructions per tree level. For a tree of 15 levels and frames containing 10 points, the algorithm computes the correspondence between two frames in 1.5 msec, including computation of the initial solution.

4.2 Discussion

This algorithm illustrates some of the same characteristics of the duplicate-and-delay algorithm shown in the MSIMD Hough. Here what is duplicated is essentially control information. The downward flow of candidate match information is conflated with a relatively inexpensive way of generating additional, though redundant, candidate matches. The one actual tree-wide computation is that for the goodness-of-fit match measure, roughly analogous to the computation of whether or not to vote in the Hough algorithm. Most communication is avoided altogether. By delaying the reporting of the distributed results until they can be compared with their siblings, the data can be easily RESOLVED Since what is required is only a maximum, it is mostly logical data that is transferred. A similar algorithm is probably possible for other fine-grained architectures, exhibiting similar functional characteristics.

5 Conclusion

In this research, we have demonstrated the feasibility of using fine-grained tree-structured SIMD machines for high-speed execution of two important intermediate-level image understanding tasks. These tasks--a Hough transform, and the correspondence portion of a moving light display algorithm--are ones for which such architectures are generally assumed not to have the advantages they possess for the more regular and communication-free algorithms of low-level signal processing. Nevertheless, these algorithms do manage to exploit the tree organization of the machine, and to reduce the effects of through-the-root communication bottleneck associated with tree architectures.

The two principal novelties of the algorithms can be summarized as the use of duplication of data (in the Hough) or of control (in the MLD), and the judicious avoidance or delay of the communication of intermediate results (in both). As ever, communication up and down the tree in these algorithms is rapid, but communication across is slow; these two strategies both help avoid lateral messages. Since the tree must be loaded anyway, the first strategy of redundancy of problem subdivision has little effect on time performance, and in fact enhances SIMD parallelism. Since one usually desires only a simple answer from an

intermediate-level vision task, the second strategy can exploit the use of the associative properties of the tree to quickly locate the best solution of the many redundant ones (as in the MLDs), or can exploit the use of the logarithmic combinatorial properties of the tree to aggregate dispersed information (as in the MSIMD version of the Hough).

NON-VON's performance on these two algorithms has been analyzed, simulated in various ways, and compared with that of other highly parallel image understanding architectures. A functional simulator, implemented on a VAX 11/750 augmented with a Grinnell image processor, has validated all of the algorithms described in this paper. A machine instruction-level simulator has also been used to execute some of the image algorithms, and to provide accurate measures of the execution time of the machine-coded versions of our algorithms. Based on these measurements, NON-VON's execution time for the two algorithms has been shown to be considerably faster than other architectures. (For details to substantiate this claim, the reader is referred to [10].)

References

1. Bacon, D., Ibrahim, H., Newman, R., Piol, A., and Sharma, S. The NON-VON PASCAL. Columbia University, May, 1982.

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

3. Brown, C. M. Peak Finding with Limited Hierarchical Memory. Proceedings of the 7th. International Conference on Pattern Recognition, Montreal, 1984.

4. Duda, R. O., Hart, P. E. "Use of the Hough Transformation To Detect Lines and Curves in Pictures." *Communications of the ACM 15*, 1 (January 1972).

5. Duff, M. J. B. A Large Scale Integrated Circuit Array Parallel Processor. Proceedings of the IEE Conference on Pattern Recognition and Image Processing, 1976, pp. 728-733.

6. Dyer, C. R. A VLSI Pyramid Machine for Hierarchical Parallel Image Processing. Proceedings of the IEEE Conference on Pattern Recognition and Image Processing, 1981, pp. 381-386.

7. Flynn, M. J. "Some Computer Organizations and Their Effectiveness." *IEEE Transactions on Computers 21*, 9 (September 1972).

8. Hough, P. V. C. Methods and Means to Recognize Complex Patterns. US Patent 3069654, 1962.

9. Ibrahim, H. A. H. Tree Machines: Architecture and Algorithms. Columbia University, June, 1983.

10. Ibrahim, H. A. H. *Image Understanding Algorithms on Fine-Grained Tree-Structured SIMD Machine*. Ph.D. Th., Columbia University, 1984.

11. Kimme, C., Ballard, D., and Sklansky, J. "Finding Circles by an Array of Accumulators." *Communications of the ACM 18*, 2 (February 1975).

12. Knuth, D. E. *The Art of Computer Programming*. Addison Wesley, 1973.

13. Kushner, T., Wu, A. U., and Rosenfeld, A. "Image Processing on ZMOB." *IEEE Transactions on Computers 31*, 10 (October 1982).

14. Merlin, P. M., and Farber, D. J. "A Parallel Mechanism for Detecting Curves in Pictures." *IEEE Transactions on Computers 24*, 1 (January 1975).

15. Potter, J. L. "Image Processing on the Massively Parallel Processor." *IEEE Computer Magazine 16*, 1 (January 1983).

16. Rashid, R.F. "Towards a System for the Interpretation of Moving Light Displays." *IEEE Transactions on Pattern Analysis and Machine Intelligence 2*, 6 (November 1980).

17. Shaw, D. E. SIMD and MSIMD Variants of the NON-VON Supercomputer Proceedings of the COMPCON Spring '84, February, 1984.

18. Shaw, D. E., and Sabety T. M. An Eight-Processor Chip for a Massively Parallel Machine. Columbia University, July, 1984.

19. Tanimoto, S. L. A Pyramidal Approach to Parallel Processing. University of Washington, January, 1983.