SIMD Tree Algorithms for Image Correlation

CUCS-222-86

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

Department of Computer Science
Columbia University
New York, N.Y. 10027
(212)280-2736

Long Paper
January 1986

Abstract

This paper examines the applicability of fine-grained tree-structured SIMD machines, which are amenable to highly efficient VLSI implementation, to image correlation which is a representative of image window-based operations. Several algorithms are presented for image shifting and correlation operations. A particular massively parallel machine called NON-VON is used for purposes of explication and performance evaluation. Although the most recent version of the NON-VON architecture also supports other interconnection topologies and execution modes, only its tree-structured communication capabilities and its SIMD mode of execution are considered in this paper. Novel algorithmic techniques are described, such as vertical pipelining, subproblem partitioning, associative matching, and data duplication that effectively exploit the massive parallelism available in fine-grained SIMD tree machines while avoiding communication bottlenecks. Simulation results are presented and compared with results obtained or forecast for other highly parallel machines. The relative advantages and limitations of the class of machines under consideration are then outlined.

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 great potential for the rapid and cost-effective execution of image understanding tasks. Although parallel architectures show considerable promise for the execution of tasks characteristic of all levels of computer vision, from low-level signal processing and the extraction of primitive geometric properties through high-level object recognition and scene analysis, our concern in the present paper will be with the class of low-level window-based tasks. In *window-based* operations, the output value of a local operation at a specific image point is a function of the image values at this point and at a number of points in its immediate neighborhood.

A number of parallel architectures [3], [10], [4], [11], [12], [13] 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. 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 [6].

This paper reports the results of investigations into the applicability of fine-grained tree-structured SIMD machines to image correlation. In order to make possible a detailed performance analysis, a number of image correlation algorithms were developed for a particular massively parallel machine, called NON-VON. The 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. While the full architecture 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.

Novel algorithmic techniques are described that effectively exploit the massive parallelism available in fine-grained SIMD tree machines while avoiding communication bottlenecks. These techniques can be summarized as vertical pipelining, subproblem decomposition, hardware-assisted associative matching, and data duplication, each of which is characterized by little or no cross-tree data flow.

In the following section, we describe the NON-VON architecture and introduce a Pascal-based parallel programming language that will be used to present the algorithms that form the central focus of this paper. In Section 3, we introduce certain hierarchical data structures for image processing and demonstrate how they can be used to represent images in a fine-grained SIMD tree machine; issues related to image I/O and system initialization are also discussed in this section. The algorithms themselves are presented and discussed in Sections 4. The final section summarizes the apparent advantages and limitations of fine-grained SIMD "pure" tree machines for the kinds of window-based

image understanding algorithms we have studied.

2 The NON-VON Architecture

The name NON-VON is used to describe a family of massively parallel machines designed to provide high performance on a number of computational tasks, with special emphasis on various artificial intelligence, database management, and other symbolic information processing applications. The general architecture [14] includes

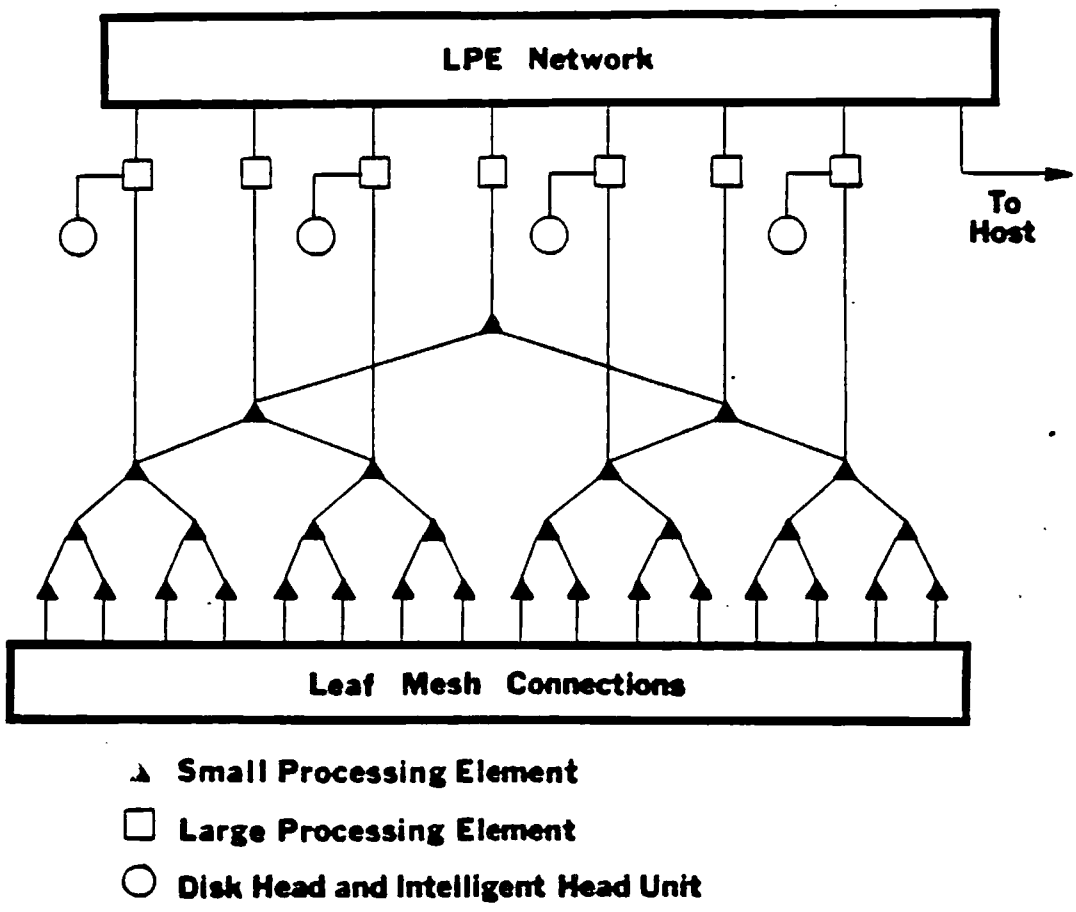
1. A massively parallel *active memory* based on a very large number (as many as a million) of *small processing elements* (SPE's).
2. A smaller number of *large processing elements* (LPE's), interconnected by a high-bandwidth interconnection network.
3. A *secondary processing subsystem* (SPS) based on a bank of intelligent disk drives.

Figure 1 depicts the top-level organization of the general NON-VON architecture. Only some of the subsystems depicted in this figure, however, are directly relevant to the concerns of this paper. In particular, although each LPE in the general architecture is able to generate a different instruction stream, supporting *multiple instruction stream, multiple data stream* (MIMD) and *multiple-SIMD* operation, the algorithms we will describe are strictly SIMD in nature, and do not require the use of multiple LPE's (or of the high-speed network used to connect them). For purposes of this paper, we will thus assume only a single LPE, which will be referred to as the *control processor* (CP).

The active memory is constructed using custom VLSI circuits, the most recently fabricated of which contains four 8-bit SPE's. 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. In order to maximize circuit yield, the initial working prototype was fabricated with chips containing only a single processing element, and incorporating only 32 bytes of local RAM; 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.

Inter-SPE 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. In the current version of the general NON-VON machine, the SPE's are configured as a complete binary tree whose leaves are also

Figure 1: Organization of the General NON-VON Machine



interconnected to form a two-dimensional orthogonal mesh. In addition, the tree may be configured to function as a linear array. Since the present paper considers only the behavior of "pure" tree machines, however, neither the mesh connections nor the machine's support for linear neighbor communication will be of concern here. Two modes of communication will be employed in the algorithms presented in this paper:

1. *Global bus communication*, supporting both broadcast by the CP to all SPE's in the active memory, 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 active memory 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.

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. If only a single SPE is enabled, the REPORT instruction may be used to transfer data from that SPE to the CP using global bus communication.

The SPS is based on a bank of "intelligent" disk drives, which are connected to all SPE's at a particular, fixed level within the active memory tree, providing a high-bandwidth path for parallel data transfers between the active memory 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/active memory interface, however, will be relevant to the concerns of this paper.

The first member of the NON-VON family, NON-VON 1, has been operational at Columbia since January, 1985. 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, and to validate the essential architectural principles underlying the NON-VON machine. NON-VON 3 [15] is based on modified chips containing four 8-bit SPE's. The NON-VON 3 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. For purposes of calculating the running time of the algorithms described in this paper, we have used the NON-VON 3 instruction set, and have assumed an execution speed of four million instructions per second [15].

While the current version of the NON-VON 3 machine does in fact incorporate mesh connections, our consideration in this paper of only the resources available in a "pure" tree machine is of interest because of tradeoffs associated with the implementation of mesh connections. From a performance perspective, the absence of mesh connections slows 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 less extensive connectivity of the strictly tree-structured topology results in lower implementation costs. In addition, "pure" tree machines permit the use of a processor embedding scheme having 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 makes it possible to exploit decreasing VLSI device dimensions without redesigning the machine by simply replacing the old active memory chips with new ones containing a larger number of SPE's. In this paper, we avoid the use of mesh connections in the interest of clarifying the strengths and limitations of "pure" tree machines.

2.1 A Descriptive High-Level Language

To describe the NON-VON algorithms presented in this paper, we will 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, which are indicated by upper-case letters in the N-PASCAL procedures that follow, refer to a set of individual variables, one element of which is found in each SPE. Operations

involving vector variables result in the simultaneous manipulation of all elements in the set. Standard PASCAL scalar variables, whose names appear in italics, reside in the CP, and represent individual data elements. Small bold letters are used for the reserved keywords of the language.

The inclusion of vector variables within N-PASCAL allows parallelism to be specified implicitly without the introduction of a special statement type. When a vector variable appears on the left hand side of an assignment statement, for example, with a scalar expression on the right hand side, the value of the scalar expression simultaneously assigned to all members of the set of data elements represented by that vector variable. If the right hand side is itself a vector variable or expression, the assignment statement results in a different local assignment within each SPE.

A given operation may be made applicable to a particular subset of the SPE's through the use of the **where** statement, which implements a form of parallel conditional statement. The **where** statement has the following syntax:

where <*conditional expression*>
  do <*statement*>
  [**elsewhere** <*statement*>]

The **where** statement causes the statement following the **do** keyword to be executed in exactly those SPE's in which the boolean expression <*conditional expression*> (which will typically involve one or more vector variables) is satisfied. If the optional **elsewhere** clause is included, the statement following the **elsewhere** keyword is executed in those SPE's in which <*conditional expression*> is *not* satisfied.

NON-VON's tree communication capabilities are supported in N-PASCAL through the use of Pascal built-in functions. Function names that start with 'N_' correspond to NON-VON machine instructions, and their parameters correspond to the arguments of these instructions.

3 Image Representation on NON-VON

In this section, we examine certain data structures used in representing images on tree-structured machines. Hierarchical data structures such as *multi-resolution pyramids*, *quadtrees*, and *regular decompositions* [16], which are frequently used for image understanding tasks on sequential machines, can be mapped naturally onto tree machines. In what follows, we present two of these hierarchical data

structures -- multi-resolution pyramids and a modified version of quadtrees called *binary image trees* -- and show how they are used to represent images on NON-VON. Issues related to initialization of the NON-VON tree and the input and output of images in NON-VON are also discussed in this section.

3.1 Multi-Resolution Pyramids on NON-VON

A multi-resolution pyramid can be defined as a sequence $\{I(L), I(L-1), ..., I(0)\}$ of images, each represented as a two dimensional array, where $I(L)$ is the original image, and $I(m-1)$ is a version of $I(m)$ at one-fourth the resolution, in a sense we will shortly make more explicit [16]. For example, if $I(m)$ represents a version of the image with resolution 64 X 64, then $I(m-1)$ represents the same image at 32 X 32 resolution. The representation of an image at successively lower levels of resolution may be thought of as a process of low-pass filtering followed by resampling; the lower resolution images retain the gross features of the original image, while details having a higher spatial frequency are lost.

Multi-resolution pyramids can also be defined in terms of trees rather than arrays. In this case, they are referred to as *pyramid trees*. A multi-resolution pyramid is defined in terms of *quartic (4-ary) trees* as follows. The leaf nodes (which together constitute the "base" of the pyramid) represent the pixels of the image at its highest resolution. Each internal node corresponds to a single pixel in a reduced resolution image, representing the 2 X 2 region defined by its four children with a single value. This value may be computed in different ways. In a gray-scale image, each parent node is typically set equal to the average of the values stored in its four children, although occasionally the averaging is weighted and includes the children's neighbors. In the case of binary images, similar computations are made; for example, the parent might be assigned the value one (the "average") only if three or more of its children have the value one, and zero otherwise.

In the NON-VON tree, the leaf level is used to store an image at its highest resolution, and the internal levels are used to represent the image at lower resolutions. Because NON-VON is a binary tree, the resolution reduction from one level to the next up in the tree is only a factor of two, and two NON-VON levels are used to obtain the same reduction as one level in the multi-resolution pyramid. This image representation scheme is useful for the implementation of multi-resolution algorithms on NON-VON.

Once an image has been loaded or otherwise processed at its highest resolution at the leaf level, a

straightforward procedure can be used to construct the multi-resolution pyramid for either gray-scale or binary images in time proportional to the height of the tree [8]. Alternatively, under usual conditions the construction of an image pyramid can be integrated with the image loading procedure, with only a small constant factor in cost over normal loading time, as we discuss below.

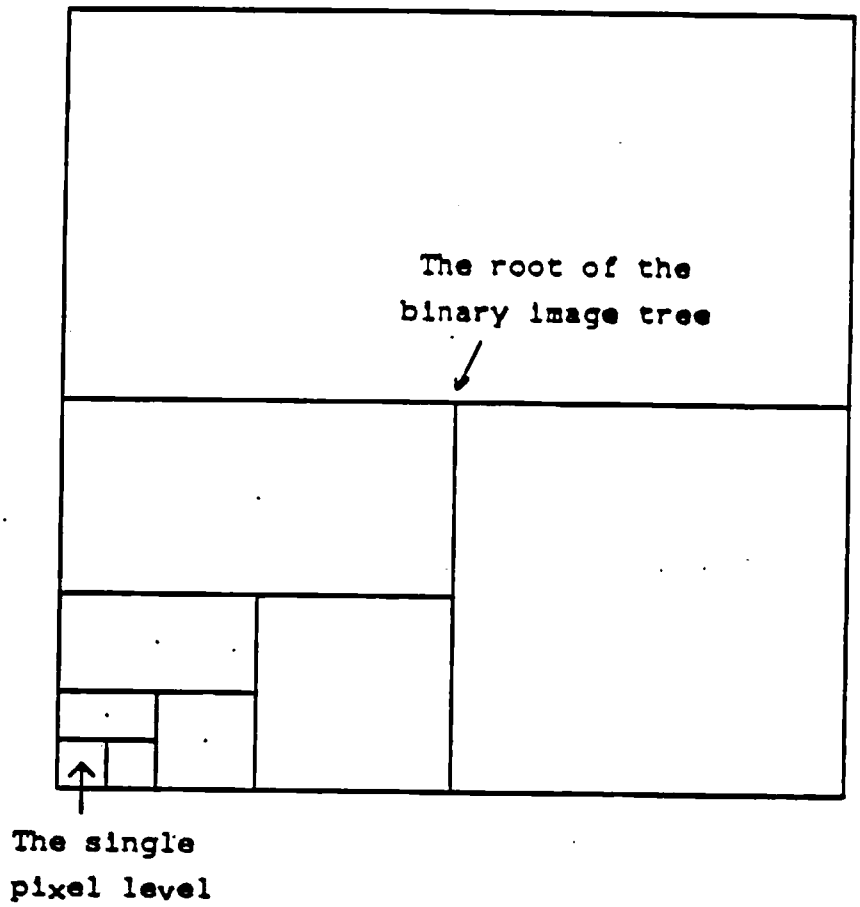
3.2 Binary Image Trees on NON-VON

The binary image tree is a variation of the quadtree data structure (which is similar in many respects to a multi-resolution pyramid) that was proposed by Knowlton [9] as an encoding scheme for compactly transmitting gray-scale and binary images. The quadtree representation of a binary image of size 2^k X 2^k, for some positive integer k, may be defined recursively as follows. If all pixels in the entire image are of the same color (either black or white), the quadtree representation of the image consists of a single node of that color. Otherwise, it consists of a single "gray" node having four children, each consisting of the quadtree representation of the four 2^{k-1} X 2^{k-1} quadrants of the original image.

Binary image trees are a variation of quadtrees in which subimages are divided into two halves at each step instead of four quadrants. Figure 2 illustrates the binary image tree data structure. Note that the shape of subdivisions changes from level to level in the binary image tree. Each subdivision is either a square, or a rectangle with the width twice the length; we will refer to subdivisions of both types as rectangles in the remainder of this paper.

Binary image trees map naturally onto a binary tree machine, and are used for all of the algorithms on binary images described in this paper. Leaf processors are used to store image information at the single pixel level, while non-leaf PE's correspond to higher-level rectangles. Each PE stores information about the location, size, contents, and adjacency relations of the part of the image it represents. A flag associated with each PE indicates whether the rectangle represented by that PE is part of the binary image tree (as it may not be, if its parent is black or white). Parallel algorithms for constructing a binary image tree and computing the information to be associated with its constituent rectangles are described in [8].

Figure 2: The Binary Image Tree



3.3 Initializing the NON-VON Tree

Prior to the execution of any of the algorithms described in this paper, the NON-VON tree is initialized to explicitly store certain image-independent spatial and control information in each PE. Note that both of the representation schemes described above use each PE to represent a rectangle of pixels from the original image; the coordinates and dimensions of this rectangle are stored within the PE.

In particular, the variables XADD and YADD used in the N-PASCAL procedures presented in the following sections are initialized to contain the horizontal and vertical coordinates, respectively, of the upper leftmost pixel in the rectangle in question (where the origin is the upper leftmost pixel in the image and the coordinate values increase to the right and downward). The variables XSIDE and YSIDE are initialized to contain the width and length, respectively, of the rectangle in pixels. The variable CURLEV is initialized to contain the level number within image hierarchy, where the root is at level 0.

This image-independent initialization procedure takes time proportional to the number of levels in the tree (19 levels in the case of a 512 X 512 image. The NON-VON 3 code for this procedure initializes a tree of 15 levels in approximately 0.3 msec [8].

3.4 Loading the NON-VON Tree

Image-dependent information is also stored within each PE during the process of image loading or pyramid construction. For a gray-scale image, the N-PASCAL integer vector variable GRAY_LEVEL is used to store the gray-scale value of the pixel or the image rectangle corresponding to each PE. In the case of a binary image, the image values are stored into the boolean vector variable BINARY. Likewise, other similar vectors would be used to hold other image-dependent features (for example, color or texture information).

Conceptually, an image is loaded or unloaded through the root of the tree, with the image pixels presented to the root in raster scan order. Since commonly available imaging devices typically present their data in this fashion anyway, no parallelization of image acquisition is necessary. Each pixel value, together with its image coordinates, is broadcast throughout the tree; only the leaf pixel with matching coordinates stores it. This can be done with or without the obvious root-to-leaf pipelining, depending on image presentation speeds. Unloading the tree can be done similarly, with the appropriate PE presenting data to

the root in response to the broadcasting of the image coordinates of the desired pixel. In this fashion, the tree acts as a simple random access memory. However, the tree can also be referenced as an associative memory, with PE's presenting to the root the image coordinates of a given pixel value. Since the components of the triple (XADD, YADD, GREY_LEVEL) are stored explicitly, this entails no added cost, and permits other exotic output modes based on standard data base retrieval operations.

Depending on its definition, usually the creation of a multi-resolution pyramid for either gray-scale or binary images can be integrated with image loading in the following way. In the most common case, non-leaf nodes of the pyramid are defined as the simple average of all its descendants. It is not difficult for each PE to compute if the broadcast image pixel is a descendant; the image coordinates of the pixel must lie within the image rectangle the PE represents. The four integers that define a PE's rectangle have been stored at tree initialization, with even leaf PE's representing image rectangles of unit width and length. This computation can be done in parallel by all PE's. Only the proper leaf PE and all its ancestors, however, will find the pixel's coordinates to be in the appropriate ranges. These selected PE's add and store the image value in an accumulator; all accumulators are then normalized in parallel at the end of image load. For grey-scale image pyramids, the normalization takes the form of division by the area of the PE's rectangle; since this area is a power of two dependent on the PE's level in the tree, it can be done by a simple shift. For binary image pyramids, this normalization is done by a comparison; in the simplest case it is a comparison against half the area of the rectangle, which also can be implemented by a shift. The cost of this integration of image load with pyramid creation is the incremental cost of doing a range comparison by each PE: a small constant factor.

In pure tree machines, loading and unloading through the root can be a bottleneck for algorithms with extensive I/O operations. In an actual NON-VON configuration, image data would be loaded and unloaded in parallel through all PE's at some intermediate level in the active memory tree. With disk units connected to the 64 PE's at the seventh level of the NON-VON tree, approximately 1.5 msec would be required to load a 512 X 512 image [8]. (Image input from sensors, however, would still be dominated by the sensor speed, unless the camera were modified to also allow such subimage partitioning.) If such parallel I/O were employed, integration of pyramid creation with loading is still possible, although the values for those PE's above the I/O intermediate level would have to created in the

straightforward sum-of-children way.

In the following sections, we assume that the image has already been loaded in the NON-VON tree, and the above N-PASCAL variables set accordingly.

4 Image Correlation

In this section, we discuss the utility of fine-grained tree-structured SIMD machines for image correlation and related image operations, known as *local* or *window-based* operations. Unlike pixel-based operations such as histogramming or thresholding where spatial location is unimportant, the output value of a local operation at a specific point is a function of the image values at this point and at a number of points in its immediate neighborhood.

The techniques and algorithms developed in this section to compute image correlation are applicable to most other local operations; consequently, performance of the image correlation algorithms is indicative of window-based operations in general. For the most part, pure tree architectures must finesse their lack of nearest-neighbor (mesh) connections. This calls for some care in decomposing problems into the recursive subproblem format that trees excel in. Occasionally, limited amounts of vertical pipelining can be exploited as well. However, as the following correlation algorithms demonstrate, without nearest neighbor connections even clever correlation algorithms do not perform very well on a pure tree.

Correlation techniques are special cases of image convolution techniques. They are widely used in many image understanding tasks, including simple filtering to detect a particular feature in an image, edge detection, image registration, motion and stereo analysis, and object detection by template matching [2]. Image correlation involves determining the location at which a relatively small *template* image best matches the input image; the goodness of the match is measured by a value of a *correlation function*.

The correlation function can be defined in many ways. One common correlation measure is the *cross_correlation*, defined for each possible relative location of the input image and the template. Let us assume an image array X and a template array Y, with x and y representing the elements of X and Y respectively. Then:

$$cross\_correlation = \sum_{i=1}^{I} x_i y_i \qquad (1)$$

On a sequential machine, the time required to execute such a function is $O(nt)$, where n is the number of pixels in the image and t is the number of pixels in the template, since the value of the cross_correlation must be evaluated with the template centered over each pixel of the image. Conceptually, the image is repeatedly shifted under the template and the sum evaluated in each possible position.

On a parallel machine, the efficient implementation of this *image shift* operation is paramount. (In fact, the mesh connections incorporated in NON-VON 3 were intended in large part to support it.) Thus, we devote the next subsection to two algorithms to perform the image shift operation on a pure tree machine, one for images represented as binary image trees, and one for those stored as a two-dimensional array in the leaves. These algorithms then form the basis for the image correlation algorithms introduced in the second subsection.

4.1 Image Shift Algorithms: Binary Image Trees

Shifting a binary image that is represented as a binary image tree can be done in two steps. First, a full, shifted binary image is created from the source binary tree; secondly, a new binary image tree is created from this new image. The first step involves reporting the size and location information of all white ("figure") rectangles, one by one, to the CP using the RESOLVE instruction. For each reported rectangle, the new location of the rectangle is computed using the reported location and the horizontal and vertical shifting required. The shifted location information and rectangle size are then rebroadcast, and all leaf PE's in parallel compute whether their pixels fall within the new rectangle boundary (including the effects of wraparound, if desired). Those that do associatively match then reset their initially black ("background') new image pixels to white. The second step--creating the new binary image tree representation of the shifted image--was described in Section 3.

This algorithm executes in time proportional to the number of white rectangles in the binary image tree representation of the image. Typically, this number is of $O(d)$, where d is the diameter of the image [5]. Thus, the time required to execute this operation is $O(n^{1/2})$, where n is the image size, since the creation of the image dominates the subsequent creation of the new image tree. Although the performance of the algorithm is dependent on the number of rectangles, it is independent of the distance to be shifted; this is the converse of the performance of a mesh-connected machine. The NON-VON 3 code executes about

55 instructions per reported rectangle [8]. Thus, shifting a 128 X 128 binary image containing 500 rectangles requires about 6.875 msec.

4.2 Image Shift Algorithms: Gray Scale Images

Next, we describe the algorithm to perform image shifting for an image stored as a two-dimensional array on the leaf level. For the sake of simplicity, we consider the case of shifting the gray-scale image one position in the left direction. Slightly modified versions of this algorithm may be used to shift the gray-scale image one position in the three other directions.

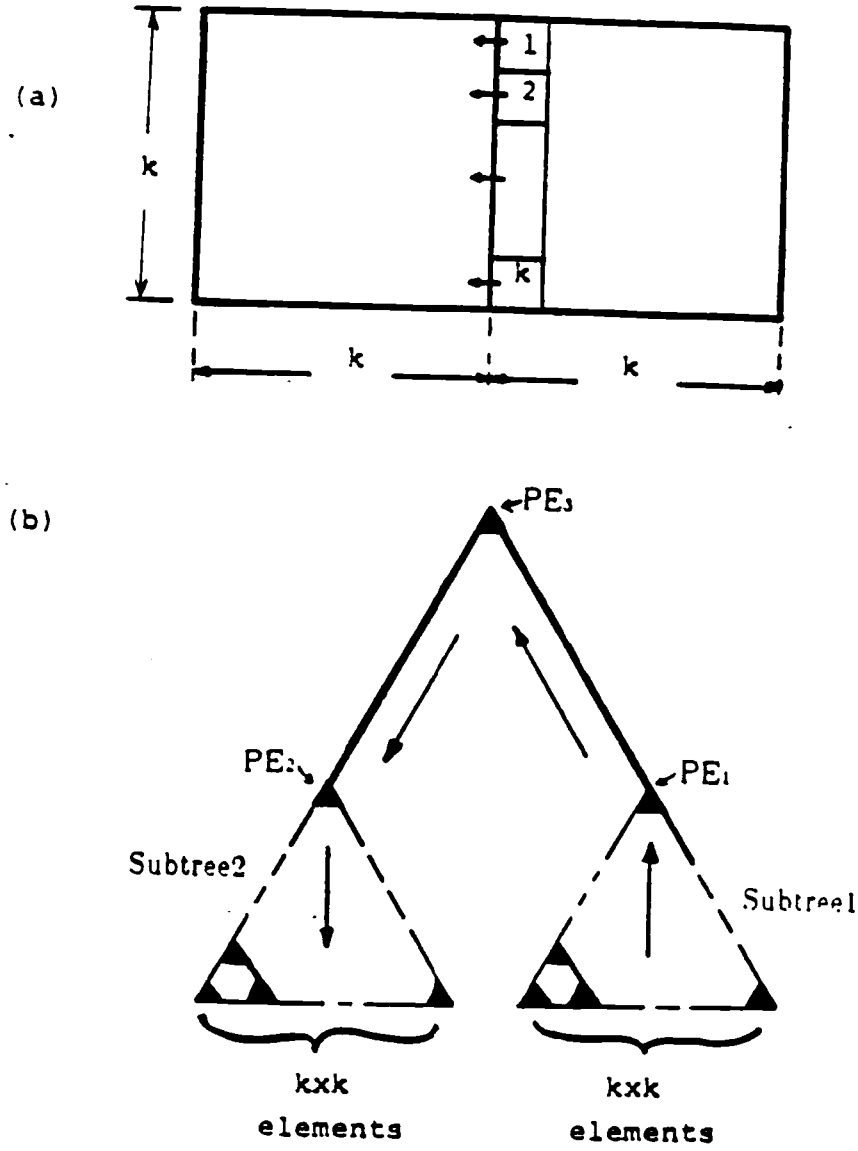
Recall that gray-scale images are stored in the leaf PE's, and that the leaf PE's of a subtree in the NON-VON tree correspond to a rectangle of the stored image. Figure 2 shows two adjacent k x k squares of the image and the NON-VON tree representation of these two subimages.

Shifting the image one position in the left direction involves a sequence of steps of increasing complexity and decreasing efficiency. In the first step, half the pixels of the image can be easily left shifted in parallel since they occupy the right leaves of a parent; they are transferred through it to the left leaf. In the second step, those leaves which are on the leftmost boundary of 2 X 2 squares can be transferred through their great grand parents to the rightmost boundary of their left neighboring 2 X 2 squares. However, full parallelism is no longer possible because two values must pass through each great grandparent.

Similarly, as shown in the figure, for any k which is a power of two, the leftmost k image values from all k x k right subtrees can be transferred to the righmost side of their corresponding left subtrees through the common roots of the two subtrees (PE3). Parallelism is degraded further, since k values must pass through a common root; however, all k x k subtrees still are ganged together in SIMD fashion. The last step transfers $n^{1/2}/2$ pixels through each of the two sons of the full tree's root; since they must be transferred one at a time, this number gives a trivial lower bound on the complexity of any left shift algorithm. This bound can actually be obtained by a limited form of vertical pipelining.

The basic procedure to transfer k elements sends them up the right subtree one by one in a pipelined fashion. Since the k values all lie along the leftmost border of the subimage square, they all have relative

Figure 3: A $2k \times k$ Subimage and its NON-VON Tree Representation



subimage x coordinates of zero, and relative subimage y coordinates in the range *0* to *k-1*. The pipelining is therefore ordered by their relative y coordinate. After a number of steps equal to the height of the right subtree, the first element reaches its root, PE1. This element is then transferred to the root of the left subtree, PE2, through the common root of the two subtrees, PE3. The algorithm continues to send the elements as they arrive in PE1 through PE3 to PE2, and moves the elements that have arrived to PE2 one level down the left subtree.

Thus, in time proportional to (k + log k), image elements on the boundary of k x k subimages are shifted one position left. Shifting the whole image requires repeating this operation for k = 1, 2, 4, ... , $(n^{1/2})/2$, where n is the image size. The time required to shift the whole image is proportional to the sum 1 + 2 + 4 + ... + $(n^{1/2})/2$, which is equal to $n^{1/2}$ - 1. Thus, the time required to shift the whole image one position left is of $O(n^{1/2})$; the algorithm is therefore optimal to within a constant factor.

The N-PASCAL algorithm to perform the shifting of k elements on the leftmost boundary of all k x k right subimages to the rightmost boundary of their neighboring k x k left subimages follows. The main routine, given last, calls five other ancillary routines which separately: select an element, pipeline it up, pass it through the common root, pipeline it down, and deposit it correctly.

```
Procedure subimage_left_shift(k, h: integer);
var
  i, j: integer;
vector_var
  RELATIVE_X, RELATIVE_Y, SHIFT_UP: integer;
  SHIFT_LC, SHIFT_RC, SHIFT_DOWN, NEW_VAL: integer;
  LEAF: boolean;
/* The following procedure enables in the right subtrees those PE's corresponding to pixels with relative
subimage locations (0,n), and prepares the gray-scale values for transfer via SHIFT_UP. */
 Procedure pick_element(n: integer);
 begin
  SHIFT_UP := 0;
  where (LEAF = true) and
    (RELATIVE_X = 0) and (RELATIVE_Y = n) do
   SHIFT_UP := GRAY_VALUE;
 end;
/* This procedure pipelines SHIFT_UPs up the right subtrees. Since it applies to the full tree, it gets its
limited pipeline effect by vertically pipelining zeros everywhere else and ORing them at the parents. */
 Procedure move_up;
 begin
  where LEAF = false do begin
    N_RECV8(LC, SHIFT_UP, SHIFT_LC);
```

```
    N_RECV8(RC, SHIFT_UP, SHIFT_RC);
    N_OR8(SHIFT_UP, SHIFT_LC, SHIFT_RC);
   end;
 end;
```

/* The following procedure transfers the up-pipelined SHIFT_UPs in the roots of the right subtrees into the down-pipelining SHIFT_DOWNs in the roots of the left subtrees. Since it applies to the full tree, it gets its limited pipeline effect by the fact that only the proper SHIFT_DOWN values are eventually stored; all else are lost. */

```
 procedure move_around;
 begin
  N_RECV8(RC, SHIFT_UP, SHIFT_RC);
  N_SEND8(LC, SHIFT_RC, SHIFT_LC);
  SHIFT_DOWN := SHIFT_LC;
 end;
```

/* This procedure pipelines SHIFT_DOWNs down the left subtrees. Since it applies to the full tree, it gets its limited pipeline effect by vertically pipelining garbage (mostly zeroes) everywhere else. */

```
 Procedure move_down;
 begin
  N_RECV8(P, SHIFT_DOWN, SHIFT_DOWN);
 end;
```

/* The following procedure enables in the left subtrees those PE's corresponding to pixels with relative subimage locations (k-1,n), and stores the transferred value in NEW_VAL. */

```
 Procedure assign_element(n: integer);
 begin
  where (LEAF = true) and
    (RELATIVE_X = k-1) and (RELATIVE_Y = n) do
   NEW_VAL := SHIFT_DOWN;
 end;

begin
```

/* MAIN ROUTINE */ /* 1. Compute the address of each image point relative to its k x k block, and mark leaf PE's. */

```
 RELATIVE_X := XADD mod k;
 RELATIVE_Y := YADD mod k;
 mark_leaf(LEAF);
```

/* 2. Call the various procedures to pipeline the boundary elements between the two blocks. Note that h is level of the subtree roots. */

```
 for i := 1 to (k+2*h) do
  begin
   if i <= k then pick_element(i-1);
   if i <  k+h then move_up;
   if (i >= h) and (i <= h+k) then move_around;
   if i >= h then move_down;
   if i >= 2*h-1 then assign_element(i-2*h+1);
  end;
end;
```

To shift the whole gray-scale image one position, this procedure is called with values of k ranging from 1

to $n^{1/2}/2$. (For somewhat greater efficiency, the cases of k equal to 1 and 2 can be explicitly coded.) For each element shifted, 72 instructions are executed, requiring 18 μsec. For a 128 X 128 gray-scale image, 2.4 msec is needed to shift the whole image one pixel to the left. Shifting the gray-scale image more than one pixel is performed by executing this algorithm a number of times equal to the number of shifts required.

This pure tree-based algorithm is still very slow by comparison with the trivial algorithm that would be used in a machine containing physical mesh connections. In the current version of NON-VON 3, for example, where the leaves are interconnected (through one-bit ports) to form a two-dimensional orthogonal mesh, a single-pixel shift would require approximately 2 μsec in the case of a gray-scale image, and 250 nsec for a binary image.

4.3 Image Correlation Algorithms: A Naive Approach

In this subsection, we describe three algorithms to perform the cross correlation operation on the NON-VON machine. The first algorithm is a direct parallel implementation of the standard sequential machine algorithm; it exploits the limited vertical pipelining of the image shift algorithm given above, and serves as a (fairly expensive) baseline. The second and third depend upon more intricate and somewhat peculiar types of subproblem partitioning, in which two different kinds of data are redundantly stored throughout the tree in order to diminish intra-tree communication. The second method redundantly stores template information, in effect broadcasting template copies that have been "pre-shifted", so that fewer image shifts are required; it also exploits a partial vertical pipeline to do the additions quickly. A third algorithm, of more limited utility and therefore only briefly described, modifies the image loading procedure to redundantly store image information: because of the associative match properties of the tree, with little additional cost the image itself can be entirely "pre-shifted"; no image shifts are required at all.

In the first algorithm, each leaf PE stores exactly one pixel of the image, and at each step it computes and accumulates one term of the cross correlation sum for that pixel's eventual output. To compute all but the central correlation function term, each leaf PE reads the value of image points in its neighboring PE's using the shift operation described in the previous subsection, and multiplies it by the corresponding template value which has been broadcast to all PE's. Consequently, the algorithm consists of a repeated

sequence of image shift and compute steps. This sequence is repeated a number of times depending on the template size. For simplicity, we assume that the template size is 3 X 3.

```
Procedure image_corr1(temp:
    array[-1..1,-1..1] of integer);
vector_var
  SHIFT_VAL, CORR_VAL: integer;
begin
/* 1. Compute the first term of the correlation function. The array temp stores the template values, and
CORR_VAL is the accumulator. */
 CORR_VAL := temp[0,0] * GRAY_VALUE;
/* 2. Compute the rest of the correlation terms by shifting in a spiral fashion. The function shift_l(X, Y)
shifts left the image represented by the vector variable X, and stores the result image in the vector
variable Y. Shift_u, shift_d, and shift_r are defined similarly. */
 shift_l(GRAY_VALUE, SHIFT_VAL);
 CORR_VAL := CORR_VAL + temp[0,1] * SHIFT_VAL;
 shift_d(SHIFT_VAL, SHIFT_VAL);
 CORR_VAL := CORR_VAL + temp[1,1] * SHIFT_VAL;
 shift_r(SHIFT_VAL, SHIFT_VAL);
 CORR_VAL := CORR_VAL + temp[1,0] * SHIFT_VAL;
 shift_r(SHIFT_VAL, SHIFT_VAL);
 CORR_VAL := CORR_VAL + temp[1,-1] * SHIFT_VAL;
 shift_u(SHIFT_VAL, SHIFT_VAL);
 CORR_VAL := CORR_VAL + temp[0,-1] * SHIFT_VAL;
 shift_u(SHIFT_VAL, SHIFT_VAL);
 CORR_VAL := CORR_VAL + temp[-1,-1] * SHIFT_VAL;
 shift_l(SHIFT_VAL, SHIFT_VAL);
 CORR_VAL := CORR_VAL + temp[-1,0] * SHIFT_VAL;
 shift_l(SHIFT_VAL, SHIFT_VAL);
 CORR_VAL := CORR_VAL + temp[-1,1] * SHIFT_VAL;
end;
```

Note that the image shifts are order dependent, principally because diagonal shifts are not supported. If the template is an odd square (the most common case), an outwardly spiraling shift order as shown in the example is always possible, thus involving every required image position without any wasted shifts. The time required to execute the function is $O(t(s+c))$, where t is the template size, c is the time required to compute a term in the correlation function, and s is the execution time of the image shift operation. On a pure tree machine like the mesh-free NON-VON configuration considered in this paper, the image shift time is $O(n^{1/2})$ for typical images, where n is the image size. Thus, the time required can be expressed in terms of image size as $O(t(n^{1/2}+c))$. If NON-VON's mesh connections were used, the image shift operation would be performed in constant time, and the computation of image correlation would require only $O(tc)$ time.

In the procedure described above, the correlation function term is computed by first multiplying two 8-bit integers, and then performing a 32-bit integer add (for a 15-level tree). This operation takes about 30 μsec on the present version of NONVON 3. As noted earlier, the image shift operation for a 128 X 128 image takes about 2.4 msec on NON-VON when the mesh connections are not used. For a 3 X 3 template, this correlation function executes in 19.2 msec, a time dominated by the image shift time.

4.4 Image Correlation Algorithms: Template Duplication

The second approach incorporates two ideas to minimize the number of image shifts. The first idea actually sacrifices some parallelism by not computing all points' correlation in parallel. It partitions the points over time, and only computes a carefully chosen subset of points at each iteration. However, these it computes without interleaved image shifts, and with the advantages of pure parallelism for multiplies and logarithmic vertical pipelining for summation. Secondly, it recovers some of the lost parallelism by pre-shifting the template and redundantly storing these values so that many of these subsets of points can be computed without any image shifting at all. The net result is that, although the number of iterations of the algorithm as a whole increases as a function of the size of the partition, the time of the algorithm decreases since the dominant cost--the number of image shifts--decreases.

This approach logically partitions the whole image into subimages of a constant size greater than the template size; each subimage becomes the focus of correlation operations that operate only on pixels entirely within its boundary. This partitioning is strictly logical; the image representation remains the same, with one pixel stored in each PE, and with subimages represented by subtrees of equal size. However, the image representation is augmented by having each PE separately (and therefore redundantly) store pre-shifted template values. In extreme cases, each PE has one image pixel but many of these values; since pre-shifting can introduce dummy zeros, there may even be more of these values than there are values in the template itself. Though costly in space, the only need for time-demanding image shifting occurs when attempting a correlation on an image point near a subimage boundary.

The following specific example illustrates the algorithm. Figure 4 depicts a subimage of size 4 X 4 and a template of size 3 X 3. There are four central locations in this subimage at which the correlation function can be computed using only the subimage gray values: they are at the relative subimage locations 5, 6, 9,

and 10.

The computation of the correlation function for these four locations is performed by storing in every PE of the subtree four values; they are the values that the template achieves at that PE when the template is centered over each of the locations 5, 6, 9, and 10. This pre-shifting and storage can be done in parallel over all the subimages in time proportional to central area size (four) plus template size (nine), as follows. In each PE, an array indexed by relative central location is initialized with dummy zero values. Template coordinate and value triples are then broadcast one at a time, with each PE calculating which relative image location would be the template center if it were to take on the given broadcast template coordinates. Only if the relative location is one of the central ones is the template value stored opposite the appropriate central location index.

After the template is broadcast, the correlation is computed for all pixels, in every subtree, at relative location 5. All the products necessary for the correlation are formed in parallel by the 16 Pe's; seven of the products are dummies. The sum is formed using vertical pipelining to the root of the subtree, from whence it is deposited into the proper output image pixel at location 5. All correlations for pixels in relative locations 6, 9, and 10 are then computed likewise.

There are many ways to compute the correlation at the remaining border pixels. The following way is optimal, and is easily generalizable under certain assumptions; see the discussion below. To compute the correlation for relative locations 7 and 11, the whole image is shifted one position to the left, into a temporary integer vector variable. Since the values formerly at 7 and 11 are now in the central area, their necessary pixel neighborhood is entirely within the subtree. Two correlations are now computed by the subtree as if for relative locations 6 and 10; however, the respective results are deposited in relative locations 7 and 11. Shifting this temporary left-shifted image one position down, into a second temporary, makes it possible to compute the correlation for relative locations 2 and 3 in a similar way. Shifting the temporary left-shifted image one position up, into the second temporary, suffices for relative locations 14 and 15. The other six boundary points can done in mirror image fashion, starting with a right shift of the original image.

Note that only six image shifts are required in this approach, instead of the eight shifts in the standard

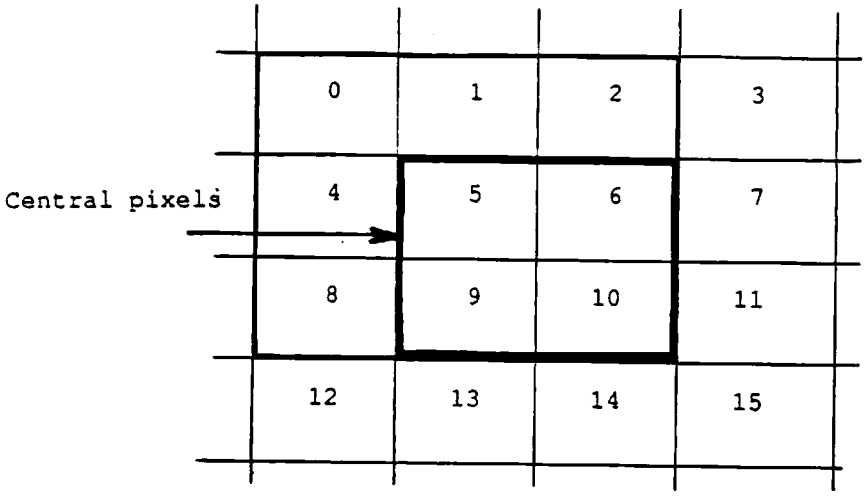


Figure 4: Image Correlation Template in a 4 X 4 Subimage

algorithm described earlier in this section. This is the minimum number of shifts possible: there are 12 border locations that must be shifted into the central area, and a unit image shift can bring at most two new image locations there. The example algorithm is therefore optimal with respect to shifts, given the subimage and template sizes. However, it should be evident from the unusual sequence of shifts and the need for temporary variables that some care was necessary to attain full shift efficiency.

We now present the example algorithm in N-PASCAL in a simplified form, and then discuss its generalization. The algorithm assumes that the PE's have stored in REL_LOC their relative location in their subtree; this is computable over the entire tree in constant time, via shifts and adds on XADD and YADD. It also assumes that the templates have been pre-loaded, so that in every PE, TEMP[*i*] is the appropriate template coefficient for calculating the correlation for relative location *i*; here, only values of *i* of 5, 6, 9, and 10 are valid indices. In practice, TEMP would be addressed by relative two-dimensional coordinates within the central area (i.e. TEMP[0..1,0..1]), at an slight incremental cost of indexing overhead. Additionally, the algorithm and the set-up code for REL_LOC and TEMP would be parameterized with respect to template size.

```
Procedure image_corr2;
var
  i, j, subtree_height: integer;
vector_var
  CORR_VAL, HOR_SHIFT_VAL, VER_SHIFT_VAL: integer;
  CORR_L, CORR_R, X, Y, REL_LOC: integer;
  TEMP: array[5..10] of integer;
  LEAF : boolean;
/* The following procedure computes the correlation function for relative location rel. The vector array
TEMP stores the template values for calculating the central locations (in this case, four of them). Since
the subimage size is 16, the root of the subimage is at level log(16); stored in subtree_height, it is used for
vertical pipeline summation and deposit of the final correlation value into location final.*/
Procedure comp_corr(rel,final: integer);
begin
 X := Y * TEMP[rel];
 for j := 1 to subtree_height do begin
   N_RECV8(LC, X, CORR_L);
   N_RECV8(RC, X, CORR_R);
   X := CORR_L + CORR_R;
  end;
 for j := 1 to subtree_height do
   N_RECV8(P, X, X);
 where REL_LOC = final do CORR_VAL := X;
end;
```

```
begin
/* 1. Compute the correlation function for points at central locations. */
 Y := GRAY_VALUE;
 for i := 5, 6, 9, 10 do comp_corr(i,i);

/* 2. Compute the rest of the correlation terms. Because of the shifts, the relative template locations are
not the same as the deposit locations. */
 shift_l(GRAY_VALUE, HOR_SHIFT_VAL);
 Y := HOR_SHIFT_VAL;
 comp_corr(7,6); comp_corr(11,10);
 shift_d(HOR_SHIFT_VAL, VER_SHIFT_VAL);
 Y := VER_SHIFT_VAL:
 comp_corr(2,5); comp_corr(3,6);
 shift_u(HOR_SHIFT_VAL, VER_SHIFT_VAL);
 Y := VER_SHIFT_VAL:
 comp_corr(14,9); comp_corr(15,10);

 shift_r(GRAY_VALUE, HOR_SHIFT_VAL);
 Y := HOR_SHIFT_VAL;
 comp_corr(4,5); comp_corr(8,9);
 shift_d(HOR_SHIFT_VAL, VER_SHIFT_VAL);
 Y := VER_SHIFT_VAL:
 comp_corr(0,5); comp_corr(1,6);
 shift_u(HOR_SHIFT_VAL, VER_SHIFT_VAL);
 Y := VER_SHIFT_VAL:
 comp_corr(12,9); comp_corr(13,10);
end;
```

The algorithm is generalizable over all template sizes, although full shift efficiency is not always possible. Assume that the template size, t, is an odd square ($t = (2b+1)^2$; b is for ''border''). Assume that the subimage area, a, is a square of a power of two ($a = w^2$; w is for ''width''); trivially, a must be greater than t. We show that the algorithm generalizes when w is chosen to be greater than or equal to $4b$. Such a choice of w is always possible, except for the unlikely case that template width is more than half the image width.

The number of central locations is given by $(w-2b)^2$; this is obtained by trimming a border of width b away from the subimage on all its sides. The remaining $w^2-(w-2b)^2$ locations are subject to image shifts. A unit image shift can bring at most $w-2b$ new image locations into the central locations. A trivial lower bound on the number of shifts is therefore given by $4b(w-b)/(w-2b)$; this is rarely an integer. (It is not hard to prove that it is an integer if and only if the central locations form a square whose side is a power of two.)

An image shift one position left enables computation in a one-pixel wide strip of w-$2b$ pixels on the rightmost border of the central locations. Left shifts can be repeated a total of b times; this enables all pixels immediately to the right of the central area. After saving this fully left-shifted image, b repeated down shifts enable computation in the right half of the pixels immediately above the central location. This is where the constraint $w >= 4b$ is necessary. It insures that the one pixel high strip of w-$2b$ pixels that are enabled after each down shift will stretch backwards from the right subimage boundary to at least the subimage center line. Starting again from the stored fully left-shifted image, b repeated up shifts enable the rest of the pixels in the right half of the subimage. The left half of the subimage is done in mirror image symmetry.

Thus, the total number of shifts is $6b$, which is always less than t-1, the number of shifts necessary for the naive algorithm. The algorithm is actually quite efficient, since if b is held constant while w grows, the trivial lower bound asymptotically approaches $4b$. Further, for some commonly employed sizes of templates--b = 1, 2, and 4--the number of shifts actually attains its lower bound. Since the number of shifts is only proportional to the square root of the template size, b, the choice of w has no effect on shift timing. It should therefore be chosen as the small as the constraint allows, in order to minimize the amount of sequential correlations and the size of the local pre-shifted template arrays.

The above N-PASCAL algorithm executes in time proportional to TEMP clear time plus template broadcast time plus shift times plus correlation times; correlations are performed once per relative location in the subimage. Thus, cost is proportional to $(w\text{-}2b)^2 + t + bs + a\log a$, where s is the unit shift time. Since w is chosen to dominate b, a dominates t; in terms of image and template sizes, this becomes $O(tn^{1/2} + a\log a)$. For moderate a, time is again dominated by shifting. In the case of 3 X 3 templates, this time is approximately 14.4 msec.

The performance figures presented above indicate that if tree communication is used to shift the whole image in any direction, then this time dominates the execution time of window-based operations on the present prototype version of NON-VON. Adding the capability of fast image shifting to NON-VON speeds up these operations considerably.

4.5 Image Correlation Algorithms: Image Duplication

The third algorithm for correlation on fine-grained tree-structured SIMD machines integrates correlation with image load in order to redundantly store image information. In limited cases, this adds only a small constant factor to image load time, but totally eliminates image shifting, yielding an algorithm whose complexity is proportional to template size. It has two drawbacks: it cannot be used on data already present in the tree, and it requires $O(t)$ amounts of local storage in each PE. However, neither of these is usually problem in practice, since correlation is usually among first operations performed, and templates are usually of moderate size with respect to PE memory capacity.

We briefly sketch the algorithm here. On image load, the triples (xadd, yadd, gray_value) are broadcast throughout the tree as before. However, each PE executes a slightly more complex associative match to test whether the incoming gray value ought to be stored. Instead of a strict comparison to its local XADD and YADD coordinates, it instead sees if the incoming point is within a half-template width of its local coordinates, and if so, it stores it within a gray value array addressed by offsets relative to the local coordinate. That is, assuming a template of size $(2b+1)^2$, the image load code is:

```
Procedure image_corr_load (xadd, yadd, gray_value: integer);
vector_var
  XADD, YADD, XOFFSET, YOFFSET: integer;
  GRAY_VALUE: array[-b..b, -b..b] of integer;
begin
  XOFFSET := xadd-XADD;
  YOFFSET := yadd-YADD;
  if abs(XOFFSET) <= b and abs(YOFFSET) <= b then
  GRAY_VALUE [XOFFSET, YOFFSET] := gray_value;
end;
```

Correlation code is straightforward:

```
Procedure image_corr3(temp: array[-b..b,-b..b] of integer);
var
  i, j: integer;
vector_var
  GRAY_VALUE: array[-b..b, -b..b] of integer;
  CORR_VAL: integer;
begin
  CORR_VAL := 0;
  for i := -b to b do
    for j := -b to b do
      CORR_VAL := CORR_VAL + temp[i,j] * GRAY_VALUE[i,j]
end;
```

This algorithm can be used in a partial form, in which only some of the image gray values that surround a point are stored in each PE; the rest are obtained in the manner of the first, naive algorithm by shifting. In fact, the first and third algorithms form a continuum, in which a varying amount of pre-shifted image information is stored locally. The first, naive method stores only the image point represented by a PE; the third redundantly stores each point's full template neighborhood; intermediate algorithms can be easily devised.

This continuum has a second dimension as well: the amount of template information pre-shifted and stored in each PE. As shown previously, the first, naive method stores no template information locally; the second method stores a subset determined by the relation of template size to subtree size. This second amount of pre-shifted information can be varied as well. It is not hard to write algorithms that redundantly store both some image points and some template points. Such redundancy appear the be the only way around the communication bottlenecks in pure tree machines.

5 Conclusion

In this paper, we have investigated the applicability of fine-grained SIMD "pure" tree machines to the execution of window-based low-level image understanding tasks. Parallel algorithms were introduced and analyzed for image shifting and correlation. Issues related to the representation of images in tree machines, and to the rapid parallel input and output of images in such machines, have also been briefly addressed. The algorithms incorporate novel techniques that exploit vertical pipelining of the tree-structured communication topology, and that reduce the effects of communication bottleneck that might otherwise be associated with the root of the tree. These algorithms also illustrate the relative disadvantage of a pure tree machine (by comparison with a mesh-connected parallel machine) in the case of such window-based operations as image correlation. This limitation may be seen even more clearly by comparing the performance of a correlation algorithm that uses only NON-VON's tree connections with one that employs its mesh connections as well. Using the mesh connections, NON-VON is able to complete the correlation task in approximately 0.5 msec -- a major improvement over the pure tree algorithm that executes in 19.2 msec for a 3 X 3 template.

One technique we have adopted to deal with the problem of communication within the tree involves

partitioning the problem into a number of smaller problems that fit within a set of independent subtrees, within each of which communication is performed locally. This approach, which is exemplified in the second image correlation algorithm described in Section 4, takes advantage of the fact that far less communication may be required between the subtrees allocated to the different subproblems than would be required if the problem were executed by the full tree in the absence of subdivision.

Performance results have been projected for the NON-VON machine (using only its tree connections, in order to address the issues of concern in this paper). Image correlation with templates of size 7 X 7 executes in 114 msec on NON-VON using only the tree part of the machine and the first algorithm. The MPP executes the same algorithm in one msec, while the estimated execution time on a VAX 11/750, on a CLIP 4, and on an ICL DAP are 3000, 16, and 16 msec respectively.

In general terms, the favorable performance and cost/performance results obtained have tended to derive from:

1. The effective use of an unusually high degree of parallelism, made possible by the machine's very fine granularity, and augmented by the redundant storage of data.
2. The extensive use of broadcast communication, content-addressable matching and other associative processing techniques.
3. The natural mapping of hierarchical and multi-resolution techniques developed by other researchers onto the machine's tree-structured physical topology.
4. The use of the tree to perform algebraically commutative and 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.

Those problems for which the limitations of SIMD pure tree machines are most apparent tend to correspond to:

1. Situations in which the root of the tree may become a significant communication bottleneck.
2. Situations in which MIMD techniques would be more effective than the SIMD approaches considered in this paper.

Although techniques have been described that may be used to minimize the impact of these limitations in certain circumstances, it would appear that the incorporation of other communication topologies and modes of instruction execution may offer significant performance and cost/performance advantages over fine-grained SIMD "pure" tree machines in certain image understanding applications. The construction

and evaluation of such machines in the context of practical image understanding problems thus remains an interesting area for future research.

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., and Brown, C. M.. *Computer Vision.* Prentice Hall, 1982.

3. 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.

4. 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.

5. Dyer, C. R. "The Space Efficiency of Quadtrees". *Computer Graphics and Image Processing 19* (1982), 335-348.

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

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

8. Knowlton. "Progressive Transmission of Gray-Scale and Binary Pictures by Simple, Efficient, and Lossless Encoding Schemes". *Proceedings of the IEEE 68*, 7 (July 1980).

9. Kruse, B. The PICAP Picture Processing Laboratory. Proceedings of the IEEE Conference on Pattern Recognition and Image Processing, 1976, pp. 875-881.

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

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

12. Reeves, A. P. "Parallel Computer Architectures for Image Processing". *Computer Graphics and Image Processing 25* (1984), 68-88.

13. Shaw, D. E. The NON-VON Supercomputer. Columbia University, August, 1982.

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

15. Tanimoto, S., and Klinger, A.. *Structured Computer Vision.* Academic Press, 1980.