# The Connected Component Algorithm on The NON-VON Supercomputer

Hussein A.H. Ibrahim

Department of Computer Science
Columbia University
NY, NY 10027

## Abstract

The NON-VON Supercomputer is a highly parallel tree-structured computer that is being implemented at Columbia University. In this paper, we demonstrate that tree architectures with their favorable characteristics for VLSI implementation, and fast global broadcast, lend themselves easily and naturally to the representation and manipulation of images represented by hierarchical data structures. A description of NON-VON architecture is presented with an emphasis on the special architectural features that will be used in our image understanding algorithms. We adopt a variation of the quadtree data structure, called the binary image tree, to represent images in the NON-VON tree. We show how images are loaded in the NON-VON tree, and present the algorithm for building the binary image trees. An efficient implementation of the connected component labeling algorithm on NON-VON is then presented. Simulation results are discussed, and we show the fast execution time of the algorithm on NON-VON. Other algorithms are also developed, such as histogramming, Hough transform, Set operations and image correlation, and we can conclude that NON-VON can be used to implement efficiently several important image understanding tasks.

---

This research was supported in part by the Defense Advanced Research Agency under contract 5-2001 / N00039-82-C-0427.

# Table of Contents

# List of Figures

# 1. Introduction

Image understanding tasks usually involve computations that can be performed simultaneously on hundreds of thousands of picture elements. Conventional von Neumann machines, where data elements have to be fetched sequentially for processing, are inadequate to execute such tasks efficiently. Thus parallel computers, with processing and memory elements intermingled, are highly desirable for efficient execution of image understanding tasks. Hardware implementation of highly parallel computers has been made feasible by recent advances in very large scale integrated (VLSI) circuitry. Various kinds of parallel architectures have been proposed in the literature and some of them are in various stages of implementation. Some examples are [Brow79], [Schw80], [Hill81], and [Shaw82].

There have been also several proposals for special-purpose computer architectures for image understanding systems. Cellular logic arrays, proposed by Unger [Unge58] for use as parallel image processors, were the basis for many later architectural proposals. Some examples of these architectures include CLIP4 [Duff76], PICAP [Krus76], BAP [Reev80], and MPP [Pott83]. For more information regarding cellular arrays the interested reader is referred to [Rose83]. Other architectures proposed for image understanding make use of pipelining as a way of introducing parallelism in the system [Kush82], or use a high-bandwidth interconnection network for communications between the PE's [Sieg81]. Hierarchical architectures for image understanding systems (referred to in the literature as hierarchical, cone, or pyramid machines) are also proposed for image understanding tasks ([Hans78], [Uhr72],[Dyer81], and [Tani83]), and they are attracting considerable attention because of their desirable characteristics for VLSI implementation [Mead79].

The NON-VON (non von Neumann) supercomputer [Shaw82], currently being built at Columbia University, is such a hierarchical machine. Its architecture includes a large number of small PE's placed at the nodes of a complete binary tree.

We have been able to demonstrate that several important image understanding tasks can be implemented efficiently on NON-VON. In this paper we will show how to implement the connected component algorithm on NON-VON. Other algorithms can be found in [Ibra84]. In Section Two, we will describe briefly the architecture of NON-VON, and in Section Three we will show how to represent images in its tree. In Section Four, we will present the algorithm for connected component labeling, and we will show some simulation results.

# 2. The NON-VON Supercomputer Architecture

The NON-VON Supercomputer [Shaw 82] is currently being implemented at Columbia University. Its architecture includes a tree-structured Primary Processing Subsystem (PPS) based on custom nMOS VLSI circuits, along with a Secondary Processing Subsystem (SPS) based on a bank of intelligent disk drives. Figure 2-1 shows the top level organization of the NON-VON machine.

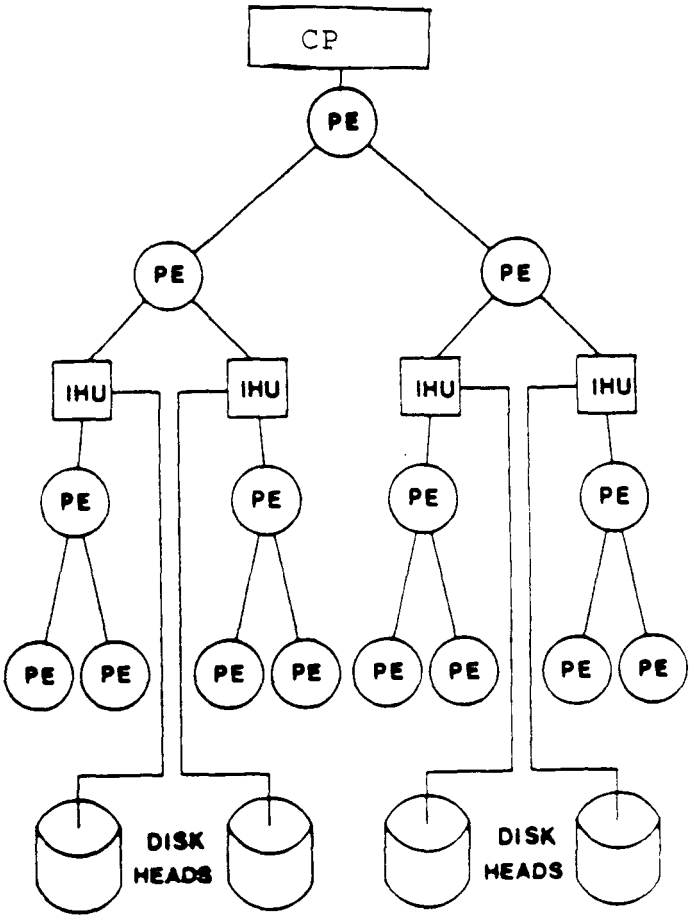



**Figure 2-1:** Top Level Organization of NON-VON

The PPS is configured as a binary tree of PE's. Each PE comprises a small RAM (32 bytes in the prototype), a modest amount of processing logic, and an input/output (I/O) switch. The I/O switch can be set for global bus communication, for communication between parents and children (tree neighbor communication), or to reconfigure the binary tree as a linear array of processors (linear neighbor

communication). NON-VON uses a hierarchical scheme for global broadcast, where broadcast data are buffered at each tree level and rebroadcast again. The first version of NON-VON, called NON-VON1, will contain chips with only one PE for the purpose of testing certain electrical and timing characteristics. The chip has been tested, and currently a tree containing 128 nodes is in the phase of construction. A modified version of the chip with sixteen PE's has been designed, and is to serve as the basis of the prototype planned to be implemented starting in the summer of 1984. This prototype is called NON-VON3. The modified chip has less area per PE, and the instruction set is made more powerful by generalizing register-to-register data transfers and adding more arithmetic processing power. It is expected that the time needed to broadcast a NON-VON3 instruction to all PE's in a tree of 20 levels (one million PE's) will be about 400 nanoseconds.

At the root of the tree is a special processor called the control processor (CP), which is responsible for coordinating different activities within the PPS. The CP is capable of broadcasting instructions to be executed in all active PE's simultaneously. This is referred to in the literature as single instruction stream, multiple data stream (SIMD) execution [Flyn 72]. Algorithms that use this mode of execution are called SIMD algorithms.

A PE actively executes the instructions broadcast by the CP as long as its *enable bit* is set. If the enable bit is reset, then the PE will be disabled and only an ENABLE instruction will activate it again. Each PE is capable of performing simple arithmetic and logical operations.

The NON-VON architecture incorporates an SPS based on a number of rotating storage devices. Associated with each disk head in the SPS is a separate sense amplifier and a small amount of logic capable of dynamically examining the data passing beneath it. Intelligent Head Units (IHU's) would be capable of performing general computations, and of serving as control processors. This will support parallel transfer of data between the PPS and SPS which is necessary to avoid the I/O becoming a bottleneck, and allow NON-VON to function as an independent collection of SIMD machines (this execution mode has since come to be referred to as multiple SIMD, or MSIMD).

An instruction called RESOLVE can be used to disable all but a single PE chosen among a specified set of PE's. This is an example of a hardware *multiple match resolution* scheme, in the terminology of the literature of associative processors. (The CP, on executing a RESOLVE instruction, is able to determine whether the operation resulted in any PE being enabled or not). The REPORT instruction transfers data from the single chosen PE to the CP using the global bus communication.

There are mechanisms by which we are able to enable only leaf PE's, or only the root PE [Shaw83]. In the following sections, variables that are stored in the tree PE's, will be referred to as local variables, while global variables will refer to those that are stored in the CP.

# 3. Image Representation on NON-VON

In this section, we will show how two of the hierarchical data structures that are used frequently in image understanding tasks on sequential machines can be used to represent image data in the NON-VON tree. Also, a procedure for initializing the NON-VON tree, loading the image, and building the image representation will be described. Hierarchical data structures are used in image understanding tasks because they allow many algorithms to be expressed in a form suitable for divide-and-conquer techniques. They are also used by algorithms employing several levels of resolution during their execution. Hierarchical data structures include multi-resolution pyramids, quadtrees, and regular decomposition. Actually these terms are often used inconsistently in the literature to refer to each other. In what follows, we present two of these hierarchical data structures, namely multi-resolution pyramids and quadtrees, and show how they are used to represent images on NON-VON.

### 3.0.1 Multi-Resolution Pyramids on NON-VON

A multi-resolution pyramid can be defined as a sequence $\{I(L), I(L-1), \ldots, I(0)\}$ 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 half the resolution. (This is the same definition Tanimoto used in [Tani80].) The pyramid provides reduced resolution versions of the image. A multi-resolution pyramid can also be defined in terms of trees, where the leaves represent the pixels of the original image, and subsequent levels represent different resolutions of the image. An image at a specific level can be computed from the image at the level below it in the tree in different ways. Typically,a parent node is set equal to the average value of its four children. Note that the four children represent a 2 X 2 region in the image. In the NON-VON tree, the leaf level will be used to store the original image, whereas the internal levels will be used to represent the image at different 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 have the same reduction as one level in the multi-resolution pyramid. We will use this image representation whenever we deal with grey scale images. In the section on initialization and loading we will show how this can be done in detail.

### 3.0.2 Binary Image Trees on NON-VON

Quadtree data structures are similar in many aspects to the multi-resolution pyramids. They are used to encode binary images, and the nodes in a quadtree are interpreted differently from the nodes in a multi-resolution pyramid tree. A good way to visualize the quadtree is by assuming that the image is a square whose

dimension is a power of 2 ($2^n$), the quadtree data structure is built by subdividing the whole image into four square quadrants with dimensions that are half that of the image. This process is repeated recursively for each quadrant n times, until the single pixel level is reached. The resulting data structure can be represented as a *quartic* tree or a *quadtree*. The root of the tree corresponds to the whole image, the leaves correspond to the single pixels, and the nodes of the tree correspond to quadrants of the square represented by their parent node. In the case of binary images, nodes of the quadtree can take one of three values. If the node children are all black, then the node is black. If they are all white, then the node is white. The node will take the value gray if its children do not have the same value, or if they all have the value gray. All subtrees rooted with a white, or a black node can be omitted, thus reducing significantly the amount of memory required to store the picture on a sequential machine.

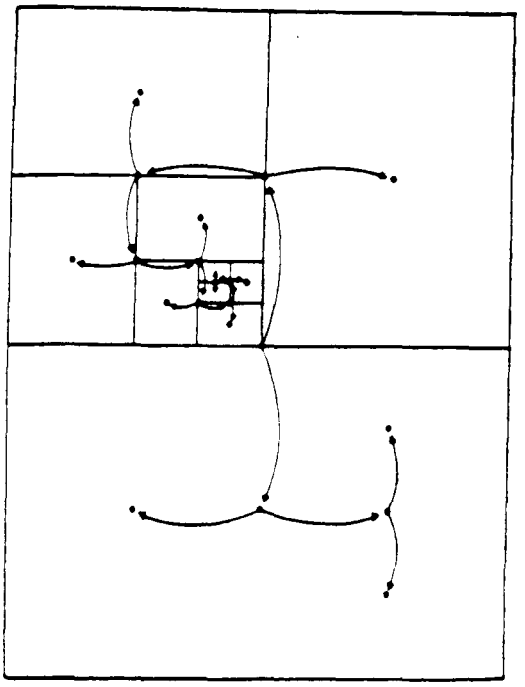


**Figure 3-1:** Binary Image Trees Block arrangement

Binary trees represent a data structure that is a variation of the quadtrees, and that has been proposed recently by Knowlton [Know80] as an encoding scheme for transmitting Gray-Scale and binary images. Figure 3-1 shows the block arrangement for this data structure. The shape of subdivisions change from level to level. It is either a square, or a rectangle with the width twice the length. One can simply determine the shape of the division at any level by testing to see if the level number is odd or even. Going from one level to the next increases the resolution by only a factor of two, while in quadtrees the resolution is increased by four. Binary

image trees are mapped naturally onto binary tree machines. On NON-VON, the leaf processors will be used to store the image information at the single pixel level, while non-leaf PE's will correspond to rectangles of pixels. A record associated with each PE will be used to store information about the location, size, and adjacency relation of the part of the image it represents.

## 3.1 Initialization and Image Loading

The connected component labeling algorithms on NON-VON, which will be described in the following sections, use information that are stored initially in each PE. Each PE corresponds to a rectangle in the original image. On the leaf PE's level, this rectangle corresponds to a single pixel in the two dimensional array that represents the image. The location of each rectangle is specified as the coordinates of its upper leftmost corner pixel. We will call the horizontal direction the x-direction, while the vertical direction will be called the y-direction. The origin of the coordinate system (0,0) is the upper left most pixel in the image, and it is increasing right in the x-direction, and down in the y-direction. Besides the x-address and y-address, each PE will store the width (x-side), and the length (y-side) of the rectangle it represents. For a 512 X 512 image, four 9-bits are needed to store the location and size information. The root level will be labeled the 0th level, while the leaf level is the nth level. Other information is also stored in each PE, and will be described later.

### 3.1.1 Initialization Procedure

The initialization procedure includes assigning to each PE in the tree a rectangle in the image, and storing in that PE the address and size information for this rectangle. We will describe informally in what follows the algorithm for initializing the NON-VON tree:

1. The initialization algorithm starts at the root level by storing zero in its x- and y-address, and 512 (PICSIZE) in its x-side and y-side variables.

2. Enable only PE's on the next level (up-down direction). The location and size variables are assigned as follows:

```
left-child x-address = parent x-address ;
left-child y-address = parent y-address ;
if parent-level is even then
 begin
 right-child x-address =  parent x-address ;
 right-child y-address =  parent y-address + parent y-side/2;
 right-child x-side = left-child x-side = parent x-side;
 right-child y-side = left-child y-side = parent y-side/2;
 end
else
 begin
 right-child x-address =  parent x-address + parent x-side/2;
 right-child y-address =  parent y-address ;
 right-child x-side = left-child x-side = parent x-side/2;
 right-child y-side = left-child y-side = parent y-side;
 end
```

3. Step number two is repeated (n times)until the leaf level PE's are initialized .

The initialization algorithm takes time proportional to the number of levels in the tree (19 levels in the case of a 512 X 512 image).

3.1.2 Loading the Image

In tree machines loading and unloading the tree only through the root can be a bottleneck for algorithms with extensive input/output operations. To overcome this, NON-VON loading and unloading is performed through a number of intelligent disk units connected to PE's on an intermediate level. This helps distribute the input/output operations. Loading an image point through the root involves first broadcasting its x- and y-coordinates, the PE with the same values for x and y and on the leaf level will be enabled. Then the image point value is broadcast to be stored in the enabled PE. Four NON-VON instructions are required to load one image point (four microseconds execution time). Loading the image through the root only will take about one second. With disk units connected to an intermediate level (for example to the level with 64 PE's), this time can be reduced significantly.

The broadcast value is stored in the integer variable TREE. In case of gray images it will take a value depending on how the gray level intensity is being digitized (typically, this value varies from 0 for white points, to 255 for black points). In case of binary images, the integer variable TREE is set to either the value 1, if the image point is black, or the value 0 if the image point is white. A character variable FQUAD (only used with binary images), will be set to 'B' if TREE is equal to 1, and to 'W' if TREE is equal to 0.

### 3.1.3 Building The Binary Image Tree

In building the binary image tree, the two variables TREE and FQUAD are set in each PE such that the value of TREE corresponds to the number of black pixels in the rectangle represented by the PE, and the value of FQUAD indicates the type of the rectangle . TREE takes the value zero when the rectangle is white, and it is equal to the area of the rectangle in the case of black rectangles. The ratio of TREE to the area of the rectangle is proportional to the gray level intensity of the rectangle. The variable FQUAD takes the value 'N' if its PE is holding a black or white rectangle that is being merged with a similar one to form a larger rectangle. In case of gray rectangles it will take the value 'G'. The algorithm for building the binary image tree proceeds as follows:

1. Enable all PE's in the level above the leaf level.

2. Let all the enabled PE's read the values of TREE and FQUAD in their children. The value of TREE in the enabled PE's will be set equal to the sum of the two variables TREE in their children. FQUAD will be set to 'G' if FQUAD in the two children are different, or if one of them is 'G'. If the two variables FQUAD in the two children are both either 'W' or 'B', then the parent FQUAD will be set to the mutual value and the FQUAD in the two children will be set to 'N'.

3. If the root is reached, stop; otherwise enable all PE's in the next level in the up direction and go to step two.

After the above algorithm is executed, the root PE will have its TREE variable set equal to the number of black pixels in the whole image, and in general each PE's TREE variable is set equal to the number of black pixels in the tree rooted by that PE. Steps two and three are repeated a number of times equal to the number of levels in the tree. Thus the algorithm takes time proportional to the height of the binary tree. Every level in the tree represents the image with a specific resolution, with the leaf level having the finest resolution and the root PE representing the whole image.

### 3.1.4 Building The Multi-Resolution Pyramid

In the case of gray images, we can build the multi-resolution pyramid in the same way as before by letting the variable TREE in every PE be the average of the two values of TREE in its two children. This averaging process acts as a low pass filter, and the images we get in the internal levels will reflect the gross characteristics of the image, while fine details will be lost due to round off errors.

# 4. Connected Component Labeling on NON-VON

Connected component labeling is a basic operation in image processing that identifies the disjoint regions of an image. The connected component algorithm will assign different labels to disjoint connected regions of a binary image represented as a binary image tree on NON-VON.

Before applying the algorithm the image objects must be separated from their background. The process of separating the objects from background is called segmentation, and there are many techniques to perform it. One technique is to use a threshold value to separate foreground points from background points. This threshold value may be be a predetermined value based on some experimental information, or may be determined by gathering certain statistics about the image (*histogramming*).

## 4.1 The Connected Component Algorithm

In describing the algorithm, we will assume that only the foreground components (black areas) are to be labeled, but the same procedure can also be applied to background component (white areas). We will now describe the algorithm informally.

The algorithm, as implemented on NON-VON, will start by assigning the label zero to all black rectangles of the binary image tree. The RESOLVE instruction is then used to report to the CP the black rectangles of the binary image one by one, in order of their sizes. This can be done easily by starting at the root level and enabling only PE's holding black rectangles at that level and then reporting them to the CP in an order that depends on how the RESOLVE instruction is implemented. This order is not important to our algorithm, as all rectangles on a specific level have the same size. When all the black rectangles on the chosen level are reported, we enable the PE's with black rectangles in the next level down the tree and repeat the reporting procedure. We end when all black rectangles in the leaf level are reported.

For each reported rectangle, the CP will assign a new label if it has not already been assigned a label. The CP will broadcast instructions to mark and label all adjacent rectangles in the different directions with the same label of the reported rectangle. If during testing of adjacency, any adjacent rectangle has already been labeled, then this adjacent rectangle and all rectangles with label equal to its label value will be assigned the label of the reported rectangle. Another black rectangle is picked as described above and the labeling procedure is repeated. The algorithm

stops after all black rectangles have been reported to the CP. During the execution of this algorithm, information about the common boundaries between rectangles will be stored locally at each node to be used later for computing some geometrical properties of different components

We will now describe the variables used by the algorithm. The algorithm uses a local variable LABEL to store the region label to which the rectangle belongs. The local bit variables TE, TN, TW, and TS are used to indicate the existence of a common boundary between a rectangle and its neighbors in the east, north, west and south directions respectively. Another local variable REPORTED is used by a rectangle to mark itself reported. The global variables NEWLABEL and COMLABEL in the CP, will be used to store a new unassigned label and to store the label to be assigned to adjacent rectangles in adjaceny testing respectively. The global variable CURLEV will be used to refer to the current level from which the algorithm picks black rectangles.

In what follows, we will describe the algorithm to label the connected components of a binary image:

1. Set the LABEL variable to 0, and the variable REPORTED to 'N' in all black rectangles. Initialize the global variables NEWLABEL and CURLEV to 0.

2. Enable all PE's holding black rectangles at level number CURLEV that have REPORTED equal to 'N'. If there are no PE's satisfying this condition and CURLEV is the leaf level, then stop; otherwise increment CURLEV and repeat step 2.

3. Pick up one of the enabled PE's using the RESOLVE instruction, and report the location, label, and dimensions of the black rectangle it represents to the CP. These information is stored in each PE during the initializtion procedure. Set REPORTED in the PE holding the reported rectangle equal to 'Y' to mark it asa reported rectangle. If the LABEL of the reported rectangle is equal to 0, then increment NEWLABEL and assign its value to the global variable COMLABEL; otherwise, set COMLABEL equal to the value LABEL of the reported rectangle.

4. Test for adjacency in the four directions one at a time. This is done by broadcasting for each direction the range in which the location of the adjacent rectangles should lie. This range is computed using the reported rectangle size and location information. Only rectangles in this range will be enabled. If any of them has a label other than zero then its value is reported to the CP. Only two rectangles at most can have

their labels equal a value other than zero, as will be proven later. All adjacent rectangles labels are set to COMLABEL. During check for adjacency, tests are performed to check for image boundary cases.

5. For each adjacent rectangle with LABEL value other than zero, we broadcast this value throughout the tree, and set the LABEL variable in all PE's holding rectangles with the same LABEL value equal to COMLABEL.

6. Goto step two.

In step four a crucial part of the algorithm's efficiency is the claim that at most two of the adjacent rectangles can have labels other than zero. To prove that, assume rectangle 3 was reported to the CP and we are looking for adjacent rectangles to it along its eastern boundaries. Assume also that rectangle 2 is adjacent to rectangle 3 in the east direction as shown in figure 4-1-a. If rectangle 2 has been labeled before, then it must be adjacent to a rectangle 1 of bigger or equal size to rectangle 3. This is because rectangles are reported to the CP in order of their size. Rectangle 2 can share a common boundary with rectangle 1 in the east, north, or south direction along the boundaries of the shaded area shown in the figure. From the way we build the binary image tree, we know that if rectangle 1 is to the east of rectangle 3 and is larger or equal to it, then the distance separating them is equal to or larger than the width of rectangle 3 (L3). Thus we conclude that if rectangle 2 is adjacent to both 1 and 3 and rectangle 2 is smaller or equal to rectangle 3, then its width (L2) is equal to L3. There is only one rectangle that can satisfy this condition as shown in Fig. 4-1-b, where its two unique positions are shown. In addition to rectangle 2 in the previous case, we can have only a second rectangle 2' that could have been labeled before because it is adjacent to a larger or equal size rectangle in either the north or south direction as shown in figure 4-1-b. Figure 4-1-c shows the third possible case, where we have two rectangles that have been labeled before and which are adjacent to rectangle 3, and to larger or equal size rectangles in the north and south direction respectively.

The same proof is valid for adjacent rectangles in the other directions.

Steps two through six are repeated a number of times equal to the number of black nodes in the image. Because of the proof, each step consists of a tightly bound number of NON-VON instructions. Thus the time the algorithm takes is proportional to the number of rectangles in the binary tree. ($O(B)$) If prior information about the adjacency for single pixels were known (for example during the broadcasting of the image), then those rectangles with adjacencies only in one direction, have not to be reported to the CP, once they are labeled.

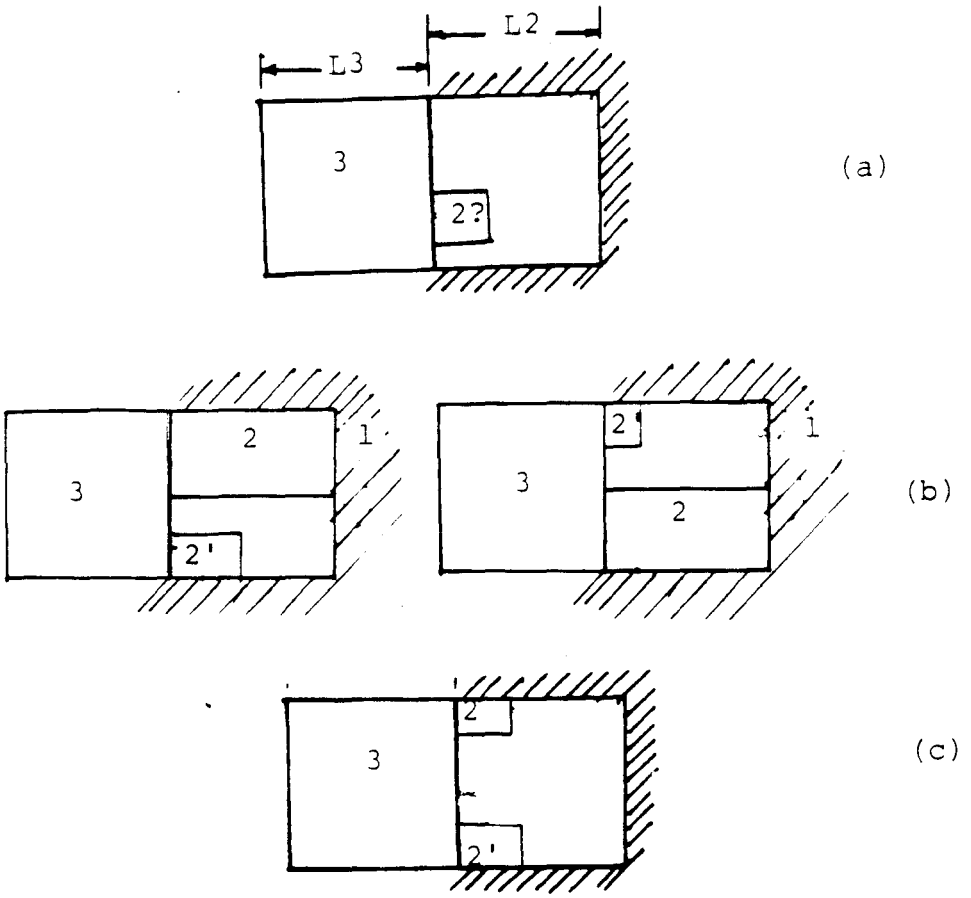



**Figure 4-1:**

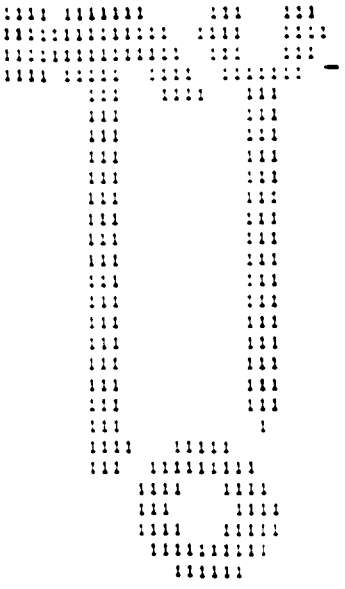
The information obtained about the common boundaries between rectangles can be used not only to compute components' properties in a time proportional to the height of the NON-VON tree, but also to mark all boundary pixels and use these information to determine adjacency relationship between components. See [Ibra84] for more information.

## 4.2 Simulation Results

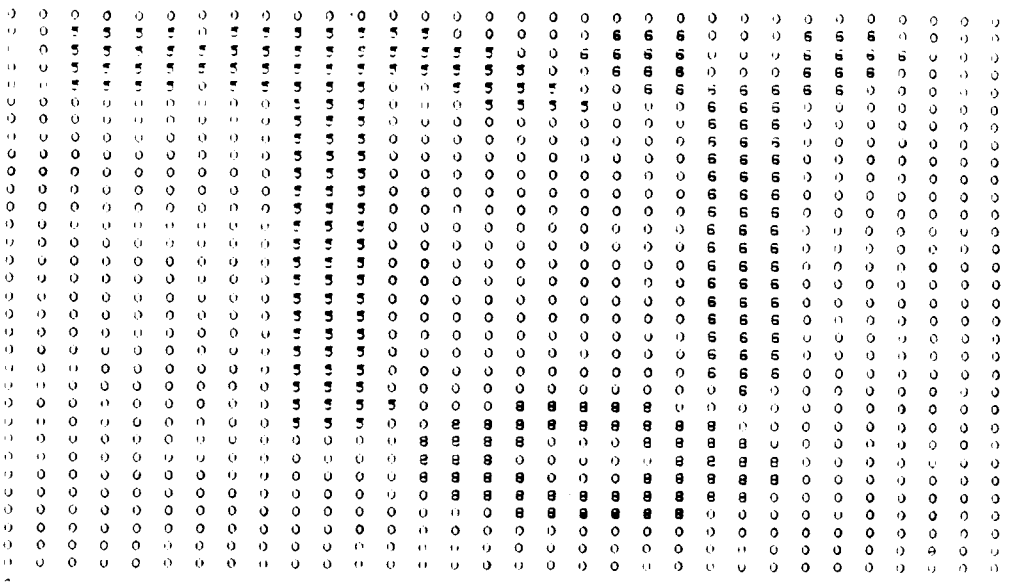
The algorithm described in the previous section was simulated on a VAX11/750 using the C programming language. Figure 4-2 shows (a) a 32 X 32 black and white image that was input to the simulator and (b) the labeled foreground components. The simulator was also used to label background objects as shown in part (c) of figure 4-2. The binary image representation of this image contains 112 black rectangles, and 146 white rectangles. It took about eight seconds for the simulator to label all black components.

The NON-VON3 code for the algorithm executes using about 180 NON-VON instructions, per iteration. With a NON-VON3 instruction cycle of 0.5 microsecond, the algorithm execution time is approximately 2B mseconds, where B is the number of black components. For a n X n binary image the average number of black

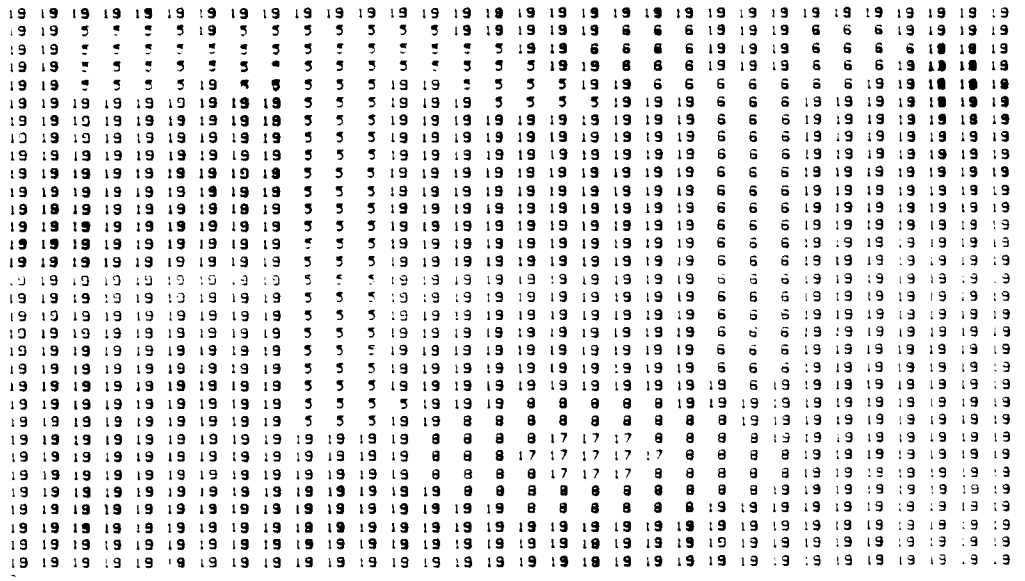
rectangles in the binary image tree is $O(n)$ [Dyer82]. Thus the average case running time for the algorithm is $O(n)$. The average running time on NON-VON for the algorithm for a 512 X 512 image is about one second. (We can always in a time proportional to the height of the tree compute the number of black rectangles in the NON-VON tree, and use that number to estimate the running time of the algorithm.)



(a) Input Black and White Image



(b) Labeled Foreground Components



(c) Labeled Foreground and background Components

**Figure 4-2:** Some Simulation Results

The simulator was also used to compute some components properties based on the information produced by the connected component algorithm (See [Ibr84].)

We can conclude that the NON-VON supercomputer can be used efficiently to implement several important image understanding tasks.

## Acknowledgments

I wish to acknowledge the excellent criticism and valuable suggestions of my advisors John Kender, and David Shaw.

# References


[Brow 79] Browning, S.
Computations on a Tree of Processors.
In *The Proceedings of The First Caltech Conference on VLSI*. January, 1979.

[Dubi 81] Dubitzki, Tsvi, Wu, A. Y., and Rosenfeld, A.
Parallel Region Property Computation By Active Quadtree Networks.
*IEEE Transactions on Pattern Analysis and Machine Intelligence* 3(6), November, 1981.

[Duff 76] Duff, M. J. B.
A Large Scale Integrated Circuit Array Parallel Processor.
In *IEEE Conference on Pattern Recognition and Image Processing*, pages 728-733. 1976.

[Dyer 81] Dyer, C. R.
A VLSI Pyramid Machine for Hierarchical Parallel Image Processing.
In *IEEE Conference on Pattern Recognition and Image Processing*, pages 381-386. 1981.

[Dyer 82] Dyer, C. R.
The Space Efficiency of Quadtrees.
*Computer Graphics and Image Processing* 19:335-348, 1982.

[Flyn 72] Flynn, M. J.
Some Computer Organizations and Their Effectiveness.
*IEEE Transactions on Computers* 21(9), September, 1972.

[Hans 80] Hanson, A. R., and Riseman, E. M.
Processing Cones: A Computational Structure for Image Analysis.
In Tanimoto, S., and Klinger, A. (editor), *Structured Computer Vision*. Academic Press, 1980.

[Hill 81] Hillis, W. D.
*The Connection Machine*.
Technical Memo, M. I. T. Artificial Intelligence Lab., September, 1981.

[Ibra 83] Ibrahim, H. A. H.
*Tree Machines: Architecture and Algorithms*
Technical Report, Columbia University, June, 1983

[Ibra 84] Ibrahim, H. A. H.
*Image Understanding Algorithms on The NON-VON Supercomputer.*
Technical Report, Columbia University, March, 1984.

[Know 80] Knowlton.
Progressive Transmission of Grey-Scale and Binary Pictures by Simple, Efficient, and Lossless Encoding Schemes.
*Proceedings of the IEEE* 68(7), July, 1980.

[Kruse 76] Kruse, B.
The PICAP Picture Processing Laboratory.
In *IEEE Conference on Pattern Recognition and Image Processing*, pages 875-881. 1976.

[Kush 82] Kushner, T., Wu, A. U., and Rosenfeld, A.
Image Processing on ZMOB.
*IEEE Transactions on Computers* 31(10), October, 1982.

[Mead 79] Mead, C and Conway, L.
*Introduction to VLSI Systems.*
Addison Wesley, 1979.

[Pott 83] Potter, J. L.
Image Processing on the Massively Parallel Processor.
*IEEE Computer Magazine* 16(1), January, 1983.

[Reev 81] Reeves, A. P.
Parallel Computer Architectures for Image Processing.
In *IEEE Parallel Processing*, pages 199-206 1981.

[Rose 83] Rosenfeld, A.
Parallel Image Processing Using Cellular Arrays.
*IEEE Computer Magazine* 16(1), January, 1983.

[Same 81] Samet, H.
Connected Component Labeling Using Quadtrees.
*Journal of the ACM* 28(3), July, 1981.

[Schw 80] Schwartz, J. T.
Ultracomputers.
*ACM Transactions on Programming Languages and Systems* 2, 1980.

[Shaw 82] Shaw, D. E.
*The NON-VON Supercomputer.*
Technical Report, Columbia University, August, 1982.

[Sieg 81] Siegel, H. J., Siegel, L. J, Kemmerer, F. C., Mueller, P. T., Smalley, H. E., and Smith, S. D.
PASM: A Partitionable SIMD/MIMD System for Image Processing and Pattern Recognition.
*IEEE Transactions on Computers* 30(12), December, 1981.

[Tani 80] Tanimoto, S.
Image Data Structures.
In Tanimoto, S., and Klinger, A. (editor), *Structured Computer Vision*. Academic Press, 1980.

[Tani 83] Tanimoto, S. L.
*A Pyramidal Approach to Parallel Processing*
Document, University of Washington, January, 1983.

[Uhr 78] Uhr, L.
Recognition Cones, and Some Test Results.
In Hanson, A. R., and Riseman, E. M. (editor), *Computer Vision Systems*. Academic Press, 1978.

[Unge 58] Unger, S.H.
A Computer Oriented towards Spatial Problems.
In *Proceedings of IRE*, pages 1744. 1958.