To the Graduate Council:

I am submitting herewith a thesis written by Mehrdad A. Saravi entitled "IMAGE PYRAMID SEGMENTATION USING PARALLEL MACHINES." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Engineering Science and Mechanics.

Dragana Brzakovic, Major Professor

We have read this thesis
and recommend its acceptance:

Donald W. Bouldin

Accepted for the Council:

Associate Vice Chancellor
and Dean of The Graduate School

## STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a Master's degree at The University of Tennessee, Knoxville, I agree that the Library shall make it available to borrowers under rules of the Library. Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgment of the source is made.

Permission for extensive quotation from or reproduction of this thesis may be granted by my major professor, or in his absence, by the Head of Interlibrary Services when, in the opinion of either, the proposed use of the material is for scholarly purposes. Any copying or use of the material in this thesis for financial gain shall not be allowed without my written permission.

Signature Mehrdad A. Sarav.

Date 12-20-91

# IMAGE PYRAMID SEGMENTATION USING PARALLEL MACHINES

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Mehrdad A. Saravi

May 1992

## ACKNOWLEDGEMENTS

The author would like to express his appreciation to Dr. D. Brzakovic, his advisor, for her support, advice, and patience during the course of this research. He would also like to thank Dr. D. Bouldin, and Dr. O. Soliman the committee members, and to his peers and friends, particularly Hamed Sari Sarraf, and Hal Beck for their encouragement and suggestions over the course of this research effort.

Lastly, and most importantly, he would like to thank his family for their help, support, encouragement and understanding when all were in great demand; projects such as this would otherwise be impossible.

# ABSTRACT

This thesis describes two algorithms: parallel fuzzy pyramid linking (FPL) image segmentation for shared memory systems and parallel FPL image segmentation for distributed memory systems. The algorithms are designed for the purpose of increasing the speed of FPL which is a specific approach to pyramid image segmentation. Both algorithms divide the original image among the available processors, and each of the processors performs FPL on each sub-image. The main disadvantage in processing time for shared memory systems is scheduling the memory usage. The main disadvantage in speed for distributed memory systems is inability of processors to share information. Both algorithms were evaluated on the process speedup. The results show that the algorithm for the shared memory systems proved to be more efficient.

# TABLE OF CONTENTS


| CHAPTER | PAGE |
|---|---|
| 1. INTRODUCTION | 2 |
| 2. PARALLEL COMPUTERS | 5 |
| 2.1 Methods Used to Improve Performance of Computers | 5 |
| 2.2 Types of Parallel Computers | 5 |
| 2.2.1 Shared Memory Systems | 9 |
| 2.2.2 Distributed Memory Systems | 14 |
| 2.3 Developing Algorithms for Parallel Computers | 22 |
| 3. PYRAMID SEGMENTATION IN PARALLEL | 26 |
| 3.1 Segmentation Using Pyramid Linking | 26 |
| 3.1.1 Pyramid Creation | 26 |
| 3.1.2 Fuzzy Pyramid Linking | 29 |
| 3.2 Pyramid Linking in Distributed Memory Systems | 33 |
| 3.2.1 Algorithm 1 (Creating Mesh on Hypercube) | 35 |
| 3.2.2 Algorithm 2 (Search for Global Minimum on Hypercube) | 38 |
| 3.2.3 Algorithm 3 (Broadcasting a Message on Hypercube) | 40 |
| 3.2.4 Embedding the Pyramid | 41 |
| 3.2.5 Fuzzy Pyramid Linking on Distributed Memory Systems | 44 |
| 3.3 Segmentation Using Shared Memory Systems | 50 |
| 3.3.1 Pyramid Creation in Shared Memory Systems | 50 |
| 3.3.2 Fuzzy Pyramid Linking in Shared Memory Systems | 54 |

**4. EXPERIMENTAL RESULTS** . . . . . 58

**5. CONCLUSIONS** . . . . . 76

**BIBLIOGRAPHY** . . . . . 77

**APPENDICES** . . . . . 80

**A. Pyramid Segmentation for Distributed Memory Systems** . . . . . 81

A.1 Host Program . . . . . 83

A.2 Node Program . . . . . 84

A.2.1 Pyramid Creation . . . . . 85

A.2.2 Finding the Mean Value of Base of Pyramid . . . . . 88

A.2.3 Linking Process . . . . . 89

A.2.4 Performing Segmentation . . . . . 95

A.2.5 Scaling the Result Image . . . . . 97

A.2.6 Fuzzy Linking . . . . . 98

**B. Pyramid Segmentation for Shared Memory Systems** . . . . . 99

B.1 Host Program . . . . . 100

B.2 Node Program . . . . . 101

B.2.1 Pyramid Creation . . . . . 102

B.2.2 Finding the Mean Value of Base of Pyramid . . . . . 105

B.2.3 Linking Process . . . . . 106

B.2.4 Performing Segmentation . . . . . 111

B.2.5 Scaling the Result Image . . . . . 113

B.2.6 Fuzzy Linking . . . . . 115

**VITA** . . . . . **116**

# LIST OF FIGURES


| FIGURE | | PAGE |
|---|---|---|
| 2.1 | A distributed memory system. | 6 |
| 2.2 | A shared memory system. | 7 |
| 2.3 | Parallel computers categorized by instructions. | 10 |
| 2.4 | Shared memory systems. | 13 |
| 2.5 | Methods of mesh topology. | 17 |
| 2.6 | Pyramid topology. | 18 |
| 2.7 | Shuffle-exchange network. | 18 |
| 2.8 | Butterfly network with 32 processors. | 19 |
| 2.9 | Hypercube of size 4. | 20 |
| 2.10 | A cube-connected cycle network with 24 processors. | 21 |
| 2.11 | An example of deadlock. | 25 |
| 3.1 | Pyramid creation. | 27 |
| 3.2 | The assignment of kernel. | 28 |
| 3.3 | Relation of nodes in pyramid. | 29 |
| 3.4 | Image extension at each level of the pyramid. | 30 |
| 3.5 | Types of links. | 31 |
| 3.6 | Pyramid segmentation on a 128 by 128 image. | 34 |
| 3.7 | Host and node processors configuration. | 35 |
| 3.8 | Mesh embedded in hypercube. | 39 |
| 3.9 | Search for global minimum value on hypercube. | 39 |
| 3.10 | Broadcasting a message in hypercube. | 41 |
| 3.11 | Base level of pyramid divided into 4 processors. | 43 |

3.12 Row interchange between the neighbor processors. . . . . 47

3.13 Construction of $t$ and $s$ in root processor. . . . . 49

3.14 Steps of pyramid segmentation in distributed memory systems. . . . . 51

3.15 Pyramid segmentation result by 4 processors. . . . . 52

3.16 Pyramid construction in distributed memory systems. . . . . 53

3.17 Steps of pyramid segmentation in shared memory systems. . . . . 57

4.1 Processor utilization for shared memory system for $N = 7$. . . . . 66

4.2 Processor utilization for distributed memory system. . . . . 67

4.3 Utilization graphs for SMS and DMS algorithms. . . . . 68

4.4 Speed up graphs for step 1 of SMS and DMS algorithms. . . . . 69

4.5 Speed up graphs for step 2 of SMS and DMS algorithms. . . . . 70

4.6 Speed up graphs for step 3 of SMS and DMS algorithms. . . . . 71

4.7 Speed up graphs for step 4 of SMS and DMS algorithms. . . . . 72

4.8 Speed up graphs for step 5 of SMS and DMS algorithms. . . . . 73

4.9 Speed up graphs for all steps of SMS and DMS algorithms. . . . . 74

# LIST OF TABLES


| TABLE | | PAGE |
|---|---|---|
| 2.1 | Methods Used to Improve Performance of Computers. | 6 |
| 2.2 | Most Common Parallel Computers. | 11 |
| 2.3 | Parallel Computers Costing Over $1M. | 12 |

# CHAPTER 1

# INTRODUCTION

Image segmentation is one of the important topics in image processing. It is defined to be the partitioning of an image into meaningful regions. Segmentation is achieved by two methods: edge based segmentation and region based segmentation. Edge-based segmentation searches for abrupt changes between adjacent regions. Region-based segmentation is done by starting with small regions, and then expanding the regions under a set of constraints. In this thesis the parallel implementation of a region based method, specifically the pyramid segmentation technique, is studied.

Pyramid segmentation is a multi-resolution approach to region growing and was first introduced by Burt [5]. Fuzzy pyramid linking is a special case of pyramid segmentation, studied by Satori-Anges [35] and implemented by Sufi [37]. An image pyramid is constructed from a base image that is the original image to be processed. The images at the subsequent levels are constructed by reducing the image by half. This reduction is done by performing convolution of images by a kernel at each level. In the case of Gaussian pyramid, this kernel approximates a Gaussian low-pass filter [5]. The structure of the image pyramid allows linking between nodes at different levels. The segmentation using fuzzy linking is an iterative process which begins with a bottom-up linking of nodes with the adjacent levels. At each iteration, the nodes at all levels except the base level nodes are replaced by new values under the linking constraints. The bottom-up process and the redefining of nodes continues until no changes in links are detected. The final step of segmentation is achieved by a top-down replacement of the value of each node by that of

the node at the adjacent level with which it has the strongest link. The study of fuzzy linking by Sufi [37] showed that fuzzy linking can improve the quality of segmentation relative to other pyramid segmentation algorithms. However, this algorithm is computationally more intensive. Parallelization of this algorithm can reduce computation time.

There have been many studies involving image processing using parallel computers that improved the performance of various algorithms as compared to using single processor computers. Atallah, et al., developed new algorithms to solve geometric problems using shared memory systems [3]. These algorithms were based on the use of a new divide-and-conquer technique. Kim, et al., designed a parallel memory algorithm that provided processors access to various parts of image data without memory conflicts [20]. A real time geometric feature extraction algorithm was designed by Liu, et al. [21]. The recent work on edge detection by Wiejak, et al., [41] was done by factorizing the convolution multidimensional masks into a sequence of one dimensional masks using Distributed Array Processecor (DAP). Willebeek-Lemair and Reeves developed a parallel method for image region growing on a multi-processor [42]. Perez designed a range data segmentation algorithm to process three dimensional images using hypercube architecture [27]. Staut used divide-and-conquer method on a variety of algorithms on meshes, trees, pyramids, and hypercubes and presented the applications for image processing algorithms [36]. In each case of these studies different methods were used to design and implement image processing algorithms to use parallel computers, and also to minimize memory management overhead problems for shared memory systems and communication overhead problems for distributed memory systems.

The central topic of this thesis is the implementation of two parallel image segmentation algorithms using fuzzy pyramid linking. One algorithm is for shared memory

systems, and the other is for distributed memory systems. These two algorithms are analyzed and compared with a single processor algorithm, according to their process speed-ups. Both algorithms divide the original image among the available processors, and each processor then constructs the subsequent level of the image pyramid in parallel. The process of bottom-up linking iteration is performed in parallel by both algorithms. The final step of segmentation, which is a top-bottom process of node replacements on sub-images, is performed by processors in parallel. The resulting subimages are then merged to construct the processed image. Both algorithms showed advantages and disadvantages. The advantage of both algorithms was speed-up of the process of image segmentation relative to a single processor. The main disadvantage for processing time in shared memory systems is scheduling the memory usage. The main disadvantage in speed for the distributed memory algorithm is the inability of processors to share information. Both algorithms were evaluated on the basis of improved processing time as compared to processing time using a single processor. The results showed that the algorithm for the shared memory system proves to be more efficient.

The organization of this thesis is as follows: Chapter 2 gives an overview of parallel computers, architectures, and methods of using parallel computers. The pyramid segmentation algorithm is described in Chapter 3. This Chapter also includes the methodologies used for parallel pyramid segmentation that are suited for shared memory and distributed memory systems. Chapter 4 compares and analyzes the performance of parallel segmentation algorithms used in shared memory systems and distributed memory systems.

# CHAPTER 2

# PARALLEL COMPUTERS

This chapter gives an overview of parallel computers. The first part of the chapter defines parallel computers and types of architectures. Thereafter, different architectures and their advantages and disadvantages are discussed. Finally methods of developing algorithms suited for parallel computers are discussed.

## 2.1 Methods Used to Improve Performance of Computers

Computer engineers have used several methods to improve the performance of computers (Table 2.1). One way to improve the performance of computers is to improve the hardware. Another way is to increase the number of concurrent operations through either parallelism or pipelining.

## 2.2 Types of Parallel Computers

Parallel computers are two or more computers that are linked to facilitate communication and to increase computing power [1,12,14,15,16,18,19,24,28,29,30,33,34,40,43]. They can be categorized by the way their memories and processors are tied in the architecture of the system. It is possible to classify the parallel computers in two distinct ways :

Table 2.1: Methods Used to Improve Performance of Computers.

| METHOD | VENDOR |
|---|---|
| Bit-Parallel Arithmetic | IBM 701 |
| Bit-Parallel Memory | IBM 704 |
| I/O Processors | IBM 709 |
| Interleaved Memory | IBM STRETCH, ATLAS, CDC 6600 |
| Instruction Look-Ahead | IBM STRETCH, IBM 360/91 |
| Multiple Functional Units | CDC 6600 |
| Instruction Pipelining | ATLAS, Amdahl 470 V/6 |
| Pipelined Functional Units | CDC 7600 |
| Data Pipelining | Cray-1 |

1. Distributive control message passing systems: Each processor controls and accesses its own memory. Interprocessor communication is conducted by a communication network using a message passing protocol (Figure 2.1). Message passing systems, also referred to as local memory systems, can be expanded to a large number of processors; however, the trade off for these systems is time loss for communication.

2. Shared memory systems: All the processors have direct access to the same memory (Figure 2.2). Current shared memory systems can be crossbar switches, multi-stage interconnection networks, and multiple bus networks. Shared memory systems are more general than message passing systems, because they have the ability to simulate message passing systems.

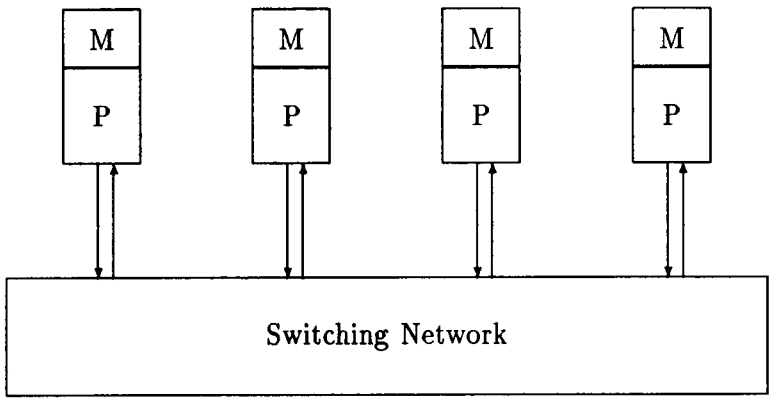



Figure 2.1: A distributed memory system.

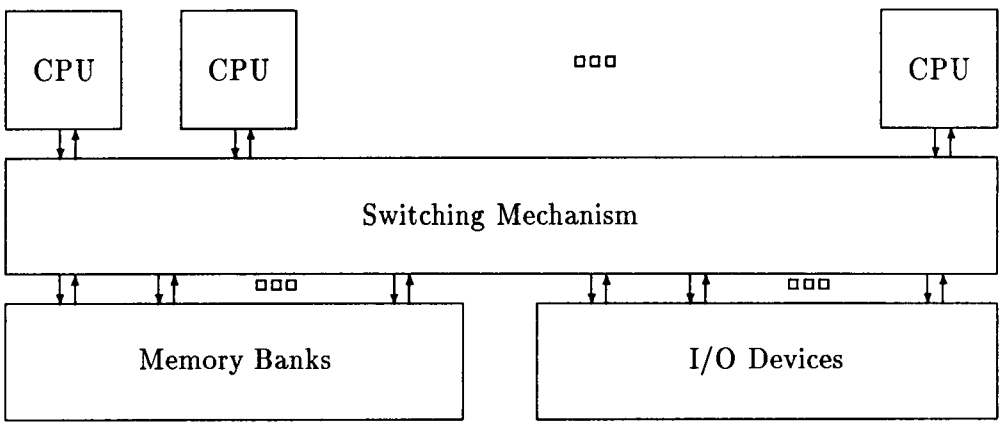



Figure 2.2: A shared memory system.

Parallel computers can also be categorized by instruction stream and data stream. An instruction stream is a sequence of instructions performed by a computer; a data stream is a sequence of data used to execute an instruction stream. Parallel computers can fall into one of the following categories [29]:

1. Single-instruction stream, single-data stream (SISD): Most computers with single processor fall into this category (Figure 2.3a).
2. Single-instruction stream, multiple-data stream (SIMD): These types of computers are also referred to as processor arrays. The execution is done by arithmetic processing units executing their own data (Figure 2.3b). The shared memory SIMD systems can also be categorized according to whether the processors can access the same memory simultaneously or not [33]:
   (a) Exclusive-read, exclusive-write (EREW): The processors cannot read from or write to the same memory location simultaneously.
   (b) Concurrent read, exclusive write (CREW): Multiple processors can read from the same memory location, but cannot write to the same memory location concurrently.
   (c) Exclusive-read, concurrent write (ERCW): Multiple processors can write to the same memory location, but cannot read from the same memory location simultaneously.
   (d) Concurrent read, concurrent write (CRCW): Multiple processors can read from and write to the same memory location simultaneously.
3. Multiple-instruction stream, single-data stream (MISD): No architecture fall into this category.

4. Multiple-instruction stream, multiple-data stream (MIMD): This is a general purpose type of multi-processing system because it enables both synchronous and asynchronous execution of multiple instructions. Most multi-processor systems fall into this category (Figure 2.3c).

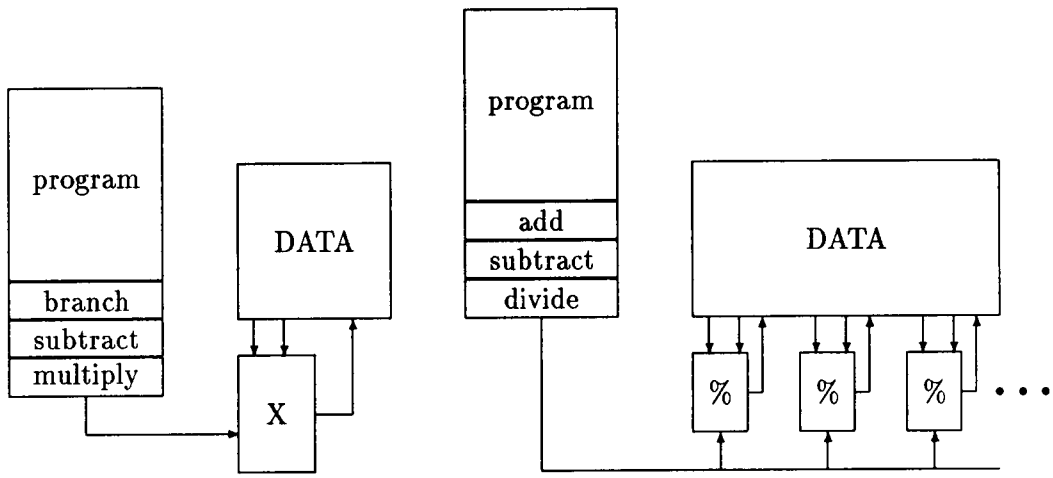
In SIMD systems, each processor executes the same instruction at every cycle, whereas in MIMD systems each processor executes instructions synchronously. Generally MIMD architectures are better suited for parallelism than SIMD because of the simpler form of programming and overall performance [24]. There are several parallel and super-computers available. Table 2.2 is a list of the most commonly used distributed and shared memory systems, and Table 2.3 is a list of the noncommercial and more expensive super-computers [13].

### 2.2.1 Shared Memory Systems

Shared memory systems are also referred to as global memory systems. The processors share the same memory; however, there is no direct connection between the processors. Shared memory systems fall into three types of architectures: cross bar switches, multi-stage interconnection networks, and multiple bus networks (Figure 2.4). Common bus interconnection limits the connection of processors to 30 [30]. Multiple buses with cross-bar switches are the extensions of a common bus. Crossbar switches allow each processor to access memory simultaneously as long as they do not access the same memory bank. Because of the high cost and complexity of designing crossbars as the number of processors grows, it is not feasible to implement crossbars for massive shared memory systems [30]. An alternative would be to implement multi-stage interconnection or orthogonal buses [19]. An interconnection network is made of switches and links that permit data

a. SISD parallel computer

b. SIMD parallel computer




c. MIMD parallel computer

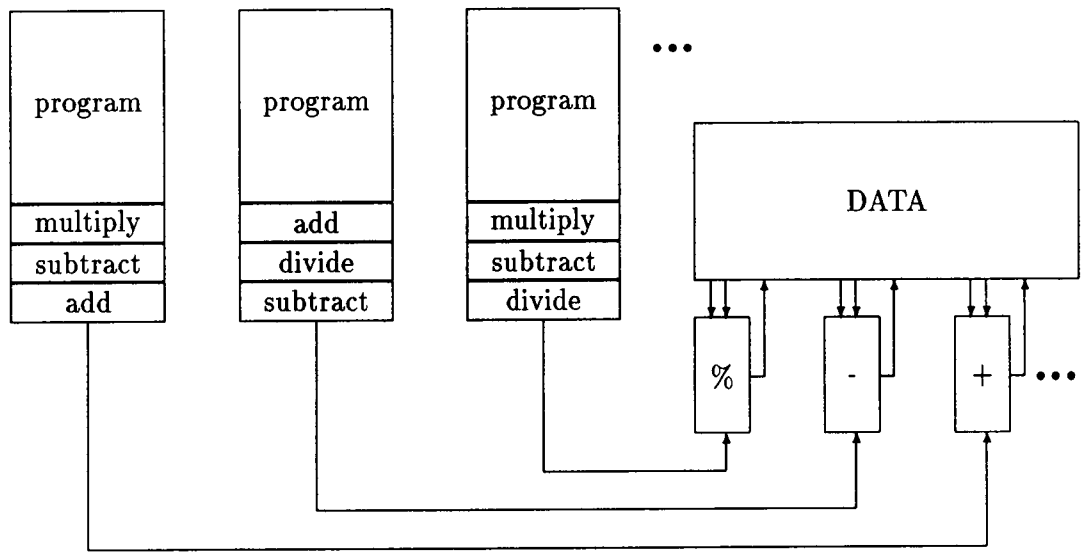


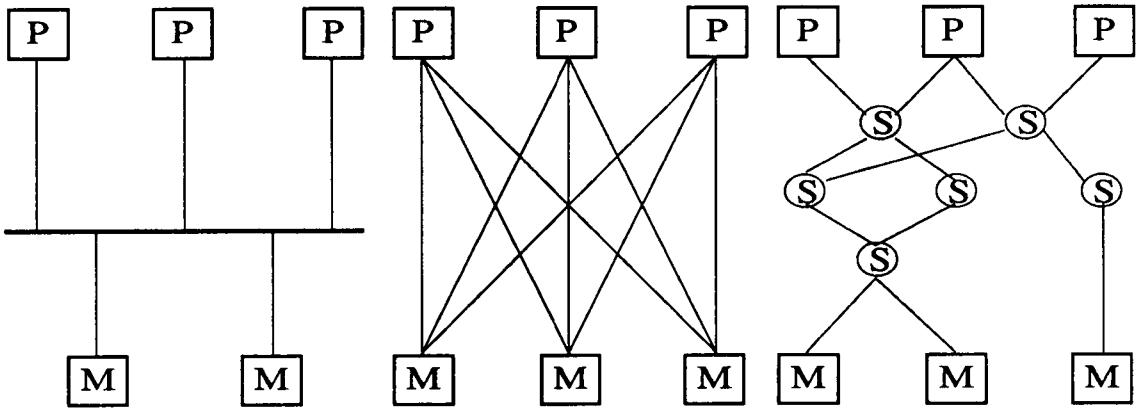

Figure 2.3: Parallel computers categorized by instructions.

Table 2.2: Most Common Parallel Computers.

| Product Model | Chip | number of processors | Parallelism |
|---|---|---|---|
| DAP | ECL | 1024 | near-neighbor |
| Ametek System 14 | 80286/80287 | 256 | hypercube |
| BBN Butterfly | 68020/68881 | 256 | Banyon network |
| TMI Connection | VLSI | 64000 | hypercube |
| Culler 7 | Gate array | 4 | bus |
| Elxsi 6400 | ECL | 12 | bus |
| Encore Multimax | 32032/32081 | 20 | bus |
| Flex 32 | 32032/32081 | 20 | bus |
| FPS-164/MAX | VLSI | 16 | bus |
| FPS-5000 | VLSI | 4 | bus |
| Intel iPSC | 80286/80287 | 128 | hypercube |
| IP-1 | N/A | 8 | cross-bar |
| Loral DATAFLOW | 32016/32081 | 256 | bus |
| Goodyear MPP | VLSI | 16384 | near-neighbor |
| Meiko | Transputer | 157 | user-configure |
| NCUBE | Custom VLSI | 1024 | hypercube |
| Sequent Balance 21000 | 32032/32081 | 30 | bus |

Table 2.3: Parallel Computers Costing Over $1M.

| Product Model | Maximum Rate in MFLOPS | Memory in Mbytes | Number of Processors |
|---|---|---|---|
| Amdahl 1400 | 1142 | 256 | 1 |
| CHoPP | N/A | N/A | 16 |
| CRAY-1 | 160 | 32 | 1 |
| CRAY X-MP | 235/proc | 128 | 1,2,4 |
| CRAY-2 | 488/proc | 2048 | 4 |
| CRAY-3 | 1000/proc | 16000 | 16 |
| CYBER 205 | 800 | 128 | 1 |
| CYBER PLUS | 100/proc | 4/proc | 256 |
| Denelcor HEP-1 | 10/PEM | 16/PEM | 16 |
| ETA-10 | 1250/proc | 2048 | 1,2,4,6,8 |
| FPS T-Series | 16/proc | 16384 | 8 - 16384 |
| Hitachi S-810/20 | 840 | 256 | 1 |
| IBM 3090/VF | 108/proc | 256 | 1,2,4 |
| Myrias 4000 | N/A | 512/Krate | 1024/Krate |
| NAS AS/91X0 | N/A | 64 | 1 or 2 |
| NEC SX-2 | 1300 | 320 | 1 |
| SAXPY | 32/proc | 512 | 32 |
| Sperry 1190/ISP | 133/proc | 64 | 1,2,4 |




a. Common bus b. Crossbar c. Multi-stage switches

Figure 2.4: Shared memory systems.

communication between processors and memory using different topologies.

Contention for any resource will reduce throughput, which means as the number of processors increases, the memory banks are accessed by more processors and will produce contention. Speed of communication between the processors and memory then becomes limited. Adding processors to a system will increase the performance of that system until a degree of saturation is reached. From this saturation point, adding more processors to the system will result in a performance decrease. The contention problem cannot be solved by adding more memory because the processors not only contend the memory banks but also block the path to memory. Even though multi-stage interconnections are effective in connecting processors, the problem for memory contention remains unsolved. Pfistor and Nordon, who were involved in developing IBMRD3 [28], showed that small non-uniformities in memory reference patterns can severely degrade the performance of

the system. This non-uniform traffic pattern occurs when all of the processors access a common variable (referred to as *hot spot access*). The memory contention can cause a traffic saturation that can block the hot spot or any other type of access. One way of resolving such a problem is to organize processors in a priority hierarchy [22]. The memory contention problem is independent of network topology.

### 2.2.2 Distributed Memory Systems

The processing elements of a distributed memory system communicate through a link network. Each node has its own local memory and there is no common global memory. Usually there is a control or host node that is more powerful than the other nodes. Systems in which all nodes are identical are called homogeneous systems. Processors in heterogeneous systems can be selected to perform particular tasks. Heterogeneous systems are suitable for a small number of nodes; however, homogeneous systems are better suited for tasks requiring a large number of nodes. Distributed memory systems which accompanied the introductions by Hoare, introduced the concept of sequential process port and proposed a simple notation for representing input/output variables in a parallel environment [18]. This was a start for researchers at CalTech to work on a homogeneous structure for message passing parallel computers. This research evolved into the Cosmic Cube project [12,34]. The Cosmic Cube is an experimental computer, built as a prototype of a VLSI implementation consisting of small computers that coordinate computations by sending messages to each other through a bi-directional, synchronous, point to point communication network. In 1985, Intel Corporation announced the iPSC, a commercial parallel computer based on the concepts introduced by the Cosmic Cube project [12,34]. Later NCUBE Corporation announced a version of the hypercube with a full custom VLSI node processor. In 1986, two other companies, Floating Point Systems

and Thinking Machines, introduced additional hypercube based systems [16].

Software and hardware determine the performance characteristics of message passing systems. From a hardware standpoint, the execution power of each node and the amount of memory available locally determines the speed at which a sequential code is executed. The band width of the communication channel as well as the communication topology strongly affects the message passing capabilities of a multi-processor system. The following factors can affect the performance of a message passing system [40]:

- Average message delay
- Message traffic density
- Total connection cost
- Number of connections per node
- Ease of routing messages
- Reliability in recovering from single-point failures
- Total number of nodes in the network
- Organization of processors or system architecture

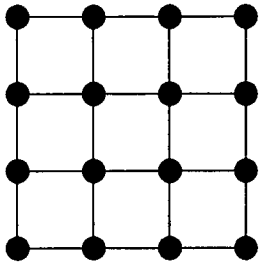
Comparative studies of message-passing capabilities [14] have determined that software plays a significant role in both sequential execution time and communication speed. Compilation strategies, operating systems, and degree of fault-tolerance embedded in the communication protocol are some of the software considerations that were determined to be of value in the total performance of a parallel system. There are six commonly used architectures for distributed memory systems [29]:

1. Mesh: One of the most flexible and powerful topologies for distributed memory systems is the regular nearest-neighbor array, or W-wide or D-dimensional mesh. Mesh connections may wrap or not wrap, and the wrap may connect the processors in columns or rows, or the connection may be toroidal (Figure 2.5). Mesh networks have disadvantages in data routing. Mesh connection is one of the suitable topologies for image processing applications. Lu, et al., used a two-dimensional mesh-connected-computer for computing several properties of a set of iso-oriented rectangles [23]. Cuerra, et al., presented two efficient algorithms to detect lines in an $n$ by $n$ image on an $n$ by $n$ mesh of processors performing a Hough transform [9].

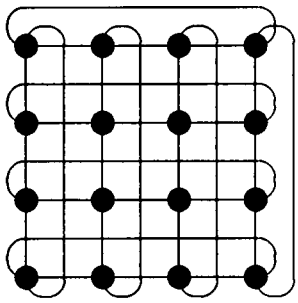
2. Pyramid: A pyramid network consists of levels of two dimensional mesh which are linked to other processors in upper and lower levels (Figure 2.6). The base mesh network of a pyramid of size $p$ has $p = k^2$ processors and the mesh network in level $l$ of the pyramid has $\frac{k^2}{2^l}$, where base level is $l = 0$. The total number of processors in a pyramid architecture is $\frac{4}{3}p - \frac{1}{3}$. Pyramid architecture has also been applied in image processing. Miller et al. [25] used pyramid architecture to determine properties of multiple objects in an image by simulating essential pyramids.

3. Shuffle-exchange network: A shuffle-exchange network consists of $n = 2^k$ nodes. The nodes are tagged from 0 to $n-1$ and have two types of connections: shuffle and exchange. Exchange connections link pairs of nodes whose tags differ in their least significant bit (Figure 2.7). The perfect shuffle connection consists of node $i$ linked to node $2i$ modulo $n - 1$, with the exception of node 0 and node $n - 1$, which have no link to other nodes.

a. no wraparound



b. row and column wraparound



c. troidal wraparound

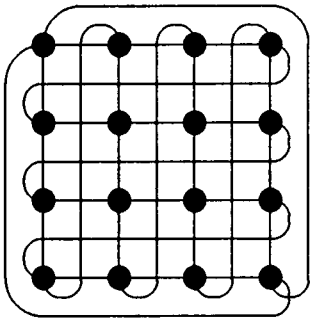


Figure 2.5: Methods of mesh topology.

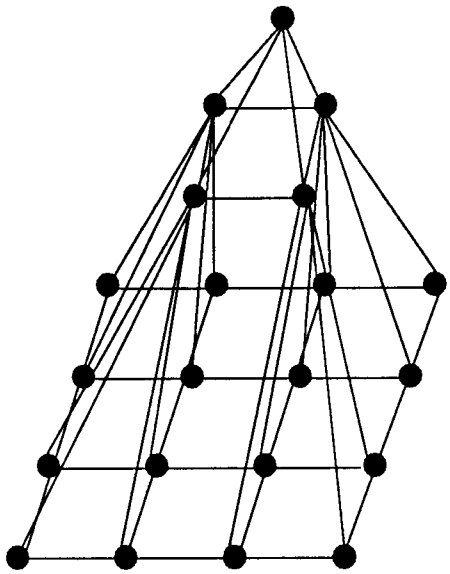


Figure 2.6: Pyramid topology.

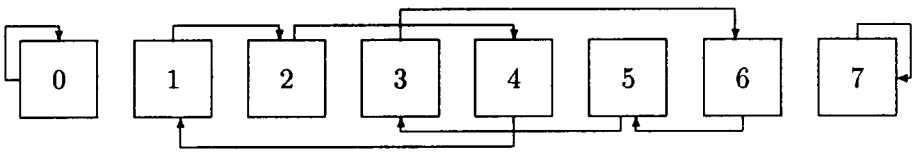



Figure 2.7: Shuffle-exchange network.

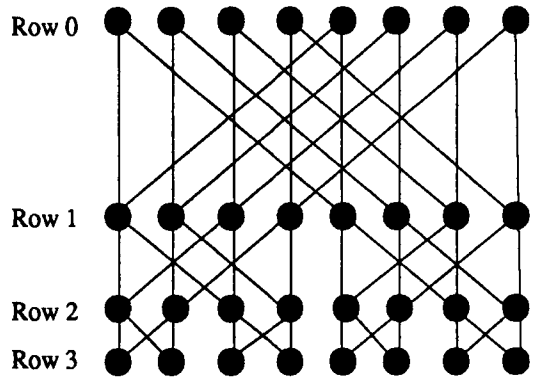



Figure 2.8: Butterfly network with 32 processors.

4. Butterfly: A butterfly network consists of $n = 2^k$ nodes. The nodes divide into $k + 1$ rows, and are labeled from 0 to $k$. Each node in the network is connected to three other nodes in different rows. For example, $p(i, j)$ is linked to two nodes on row $(i - 1)$: $p_{(i-1,j)}$ and $p_{(i-1,m)}$, where $m$ equals $j$ inverted the $i$th most significant bit in the binary representation of $j$. The connection is bi-directional. As the number of rows decreases, the number of columns increases exponentially and the network layout resembles a butterfly. Figure 2.8 shows an example of a butterfly network with 32 processors.

5. Hypercube: A cube-connected network is a graph of $2^n$ nodes, where nodes are labeled $0, 1, ..., 2^n - 1$. Two nodes are adjacent if their labels differ by one bit in binary form (Figure 2.9). There are $n!2^n$ ways in which the nodes in n-cube can be labeled [32]. A binary n-cube or hypercube with a connection graph $G$ and distance function $d(x, y)$ has the following characteristics [15]:

   - G is bipartite.
   - There are no cycles of odd length in an n-cube.
   - For each two nodes $x$ and $y$ of $G$, the number of geodesics between $x$ and

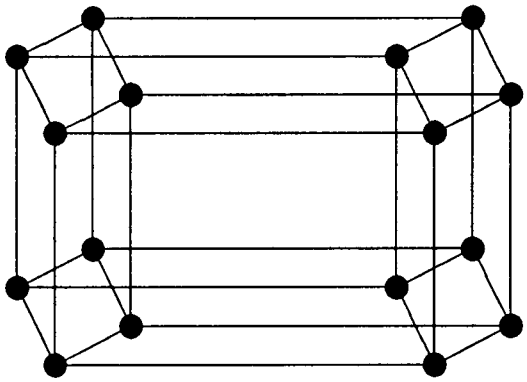


Figure 2.9: Hypercube of size 4.

$y$ is $d(x,y)$!.

- The minimum distance between two nodes $x$ and $y$ is equal to the number of bits that differ between $x$ and $y$ labeled in binary form. This is also called *the Hamming distance* $h(x,y)$.
- For each pair of nodes $x$ and $y$, there are $h(x,y)$ paths of length $h(x,y)$ and $n-h(x,y)$ paths of length $h(x,y)+2$ that do not have common nodes between $x$ and $y$.

An n-cube can be configured for other types of topologies such as meshes, rings, cylinders, etc. For example, any $m_1 \times m_2 \times ... \times m_d$, where $m_i = 2^{p_i}$ can be mapped into an n-cube where $n = p_1 + p_2 + ...p_d$ [32]. Because of the flexibility of the architecture, n-cube has been used in different applications including image processing. Mudge et al. [26] developed a general model for image processing using n-cube and by experimenting with different algorithms, it performed better than a single processor.

6. Cube-Connected Cycles: The Cube-Connected network is a hypercube of di-

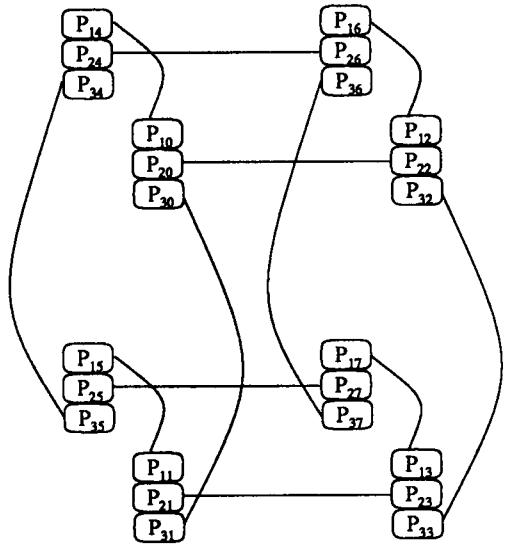



Figure 2.10: A cube-connected cycle network with 24 processors.

mension $k$, and the $2^k$ nodes are cycles of $k$ nodes that form columns of a butterfly network. For every dimension, every cycle has a node connected to a node in the neighboring cycle of that dimension. Node $p_{ij}$ is connected to $p_{im}$ if $m$ is the result of inverting the $i$th most significant bit of the binary representation of $j$. The connections are different from the butterfly network, that is, if $p_{ij}$ is connected to $p_{i-1m}$ in the butterfly network, where $j \neq m$, then $p_{ij}$ is connected to $p_{im}$ in the cube-connected network since there is a connection between $p_{im}$ to $p_{i-1m}$. Figure 2.10 shows a cube-connected network consisting of 24 nodes.

There are several message passing systems available. The following is a brief description of typical message passing systems and a history of their development:

- The Cosmic Cube: The Cosmic Cube was the first commercially available hypercube based system. The main advantage of the Cosmic Cube is that it can be scaled

up with almost no change in the design. The Cosmic Cube originated at CalTech and the design consisted of a 6 dimensional cube using Intel's 8086/8087 processors with 128 KBytes of local memory for each processor. The communication between processors takes place in 64-bit packets transmitted through synchronous full-duplex channels. A host computer performs the operations of the nodes such as communications to I/O devices. Each node has an identical operating system kernel and the user has access to the run time libraries for communication facilities.

- Intel iPSC: The Intel iPSC is the successor to CalTech's Cosmic Cube. The design can be any dimension of 2 to 7 and each processor is composed of up to 512 KBytes of memory. Communication between the processors is provided by an Ethernet chip.

## 2.3 Developing Algorithms for Parallel Computers

There are several things to consider when developing parallel algorithms. The preliminary step is to find the most suitable architecture for the algorithm. The performance of an algorithm can be affected considerably by changing the architecture. Another important factor is the degree of parallelism inherent in an algorithm, referred to as granularity [1]. Different architectures have different overheads that must be considered in performance analysis of the algorithm. For distributed memory systems the most effective overhead is communication, and for shared memory systems the most effective overhead is memory contention. Sometimes the overhead time is higher than the computation time. In general, an algorithm falls into these categories [29]:

- Pipeline algorithm: The algorithm is divided into parts, and the output of each part is the input of the subsequent part. The input of the algorithm is the input of the first part, and the output of the algorithm is the output of the last part of the algorithm. All parts of the pipeline algorithm must produce results at the same rate. If not, the algorithm must wait for the slowest part of the algorithm to complete, which can create a bottleneck for the algorithm.

- Partitioning algorithm: The problem is divided into several sub-problems and each problem is assigned to a processor. The solution of each sub-problem is part of the solution to the entire problem. The difference between a pipelining algorithm and a partitioning algorithm is that in a pipelining algorithm each processor can be doing a different type of sub-problem, while in a partitioning algorithm all processors must be processing the same type of sub-problem. The partitioning algorithm can be done in two ways:

    1. Asynchronous algorithm: The processors work toward the same goal by dividing the process into sub-processes. There are two ways to schedule the work load:

        (a) Pre-scheduling: The sub-tasks are divided and assigned to processors before execution, also referred to as static load balance [11]. This type of algorithm is best suited for a large number of sub-tasks in a complex problem.

        (b) Self scheduling: The work is not assigned to a processor until run-time, and the sub-problems are assigned to the processors by a global scheduler. Self scheduling is also referred to as dynamic load balance [11].

2. Synchronous algorithms: The processors work toward the same goal, while performing computations on different types of problems. The difference between this algorithm and the partitioning algorithm is that synchronization is not needed, which means none of the processors has to wait for another processor to provide data.

Communication and synchronization between processors is another important aspect of developing parallel programming. In order for processors to work together, they must have the ability to communicate and synchronize. Communication is obtained through shared variables in shared memory systems and through message passing in distributed memory systems. Synchronization performs two important tasks for parallel computers:

1. Constrains the ordering of events for processors to take turns accessing the message buffers for both shared memory systems and distributed memory systems. Processors cannot write to a full buffer or read from an empty buffer. To prevent these circumstances synchronization is required.

2. Controls the interference in shared memory systems. When an operation is done by all processors, it is called an *atomic operation.* A *critical section* is a sequence of atomic operations, and a mechanism must be used to secure global variables in a critical region from the other processors. Synchronization is the solution to this problem.

Synchronization is implemented differently for shared memory systems and distributed memory systems. For distributed memory systems, message passing is a form of synchronization because a message can be received only after it has been sent. Message passing can be done by either *direct naming* methods or by *global naming* methods, also

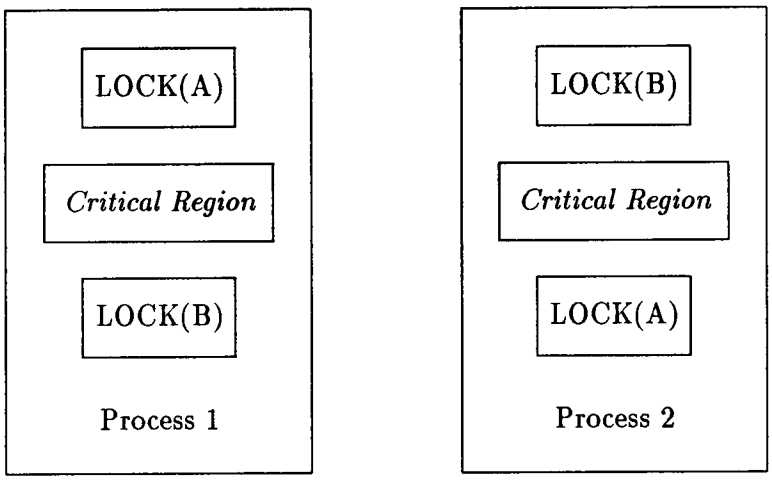



Figure 2.11: An example of deadlock.

referred to as *mailboxes* [29].

Synchronization in both shared memory and distributed memory systems can cause *deadlocks*. Figure 2.11 shows an example of a deadlock. Process A and B both lock the critical region in memory, but this lock will never be unlocked by either one of the processes.

# CHAPTER 3

# PYRAMID SEGMENTATION IN PARALLEL

This chapter involves using the parallel computers discussed in Chapter 2 for image segmentation. First, segmentation using pyramid linking and then implementation of parallel segmentation for distributed and shared memory systems is discussed.

## 3.1 Segmentation Using Pyramid Linking

Pyramid segmentation is a region-based segmentation that uses a pyramid type structure to contain the multi-resolution representations of an image. There have been numerous studies on segmentation using pyramid linking, e.g., [2,5,6,7,8,38,39]. The algorithm was introduced by Burt [5]. Segmentation using the pyramid linking approach was part of a study of texture based detection of surface defects in wood images by Brzakovic, et al. [4]. The iterative fuzzy pyramid linking algorithm for the segmentation of wood images was described by Sufi [37].

### 3.1.1 Pyramid Creation

The original image is a square array of dimension $2^N$ by $2^N$. The image is referred to as $I_0$ and is the base of the pyramid. For the consequent levels $I_1$, $I_2$ up to $I_N$, the dimension is reduced by half as level increments and resolution of the image is reduced. The top level of the pyramid $I_N$ is an array of dimension 1 by 1. A pixel or node in the pyramid at level $l$ is referred to as $I_l(i,j)$, where $i$ shows the row and $j$ shows the column

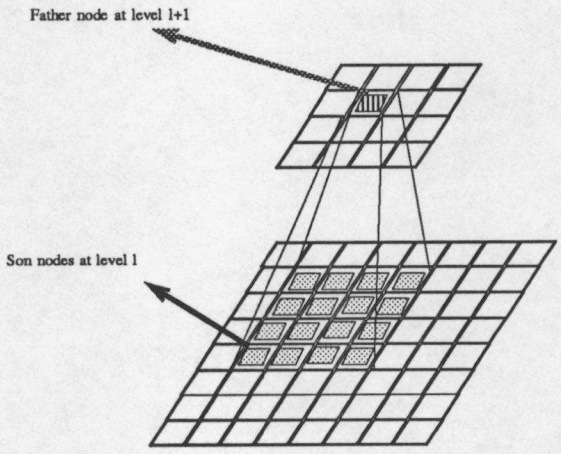



Figure 3.1: Pyramid creation.

of the image to which the node belongs. Each level of pyramid $I_l$ is created from $I_{l+1}$. These levels are $I_1$ to $I_N$ of the pyramid, and the construction is done by the overlapping average method. Each node is a weighted average of nodes at the level below in a 4 by 4 window, as shown in Figure 3.1, given by

$$I_l(i,j) = \sum_{m=-2}^{1} \sum_{n=-2}^{1} w(m,n) I_{l-1}(2i+m, 2j+n) \quad for \;\; 0 \leq i,j < N_l,$$

where $w(m,n)$ represents the discrete function for the Gaussian Kernel, and $N_l$ represents the size of the image $I_l(i,j)$, i.e., $N_l$ by $N_l$. The 4 by 4 Gaussian kernel $w$ is chosen based on the following constraints:

1. Separable: $w(m,n) = \hat{w}(m)\hat{w}(n)$
2. Normalized: $\sum w(m,n) = 1$
3. Symmetric: $w(m,n) = w(2-m,n) = w(m,2-n) = w(2-m,2-n)$

| | | | |
|---|---|---|---|
| $a^2$ | $ab$ | $ab$ | $a^2$ |
| $ab$ | $b^2$ | $b^2$ | $ab$ |
| $ab$ | $b^2$ | $b^2$ | $ab$ |
| $a^2$ | $ab$ | $ab$ | $a^2$ |

Figure 3.2: The assignment of kernel.

The combination of the constraints yields a quadratic relation between the two unknowns $a$ and $b$ for the kernel, given by

$$4b^2 + 8ab + 4a^2 = 1.0.$$

The assignment of $a$ and $b$ in the kernel is shown in Figure 3.2. The unknowns $a$ and $b$ can be chosen based on the solution of the equation for which one solution is

$$a + b = 0.5.$$

The shape of these equivalent weighing functions is such that if $a$ and $b$ are chosen to have values $a = 0.125$ and $b = 0.375$, the weighing functions then closely approximate the Gaussian probability density function. The image pyramid thus created is known as the *Gaussian pyramid.* In Gaussian pyramid, each node at level $l$ has a 4 by 4 array of *candidate son* nodes at level $l - 1$. For node $I_l(i, j)$ there are 16 *candidate son* nodes $I_{l-1}(i', j')$, where

$$i' = 2i - 2, 2i - 1, 2i, 2i + 1$$

$$j' = 2j - 2, 2j - 1, 2j, 2j + 1.$$

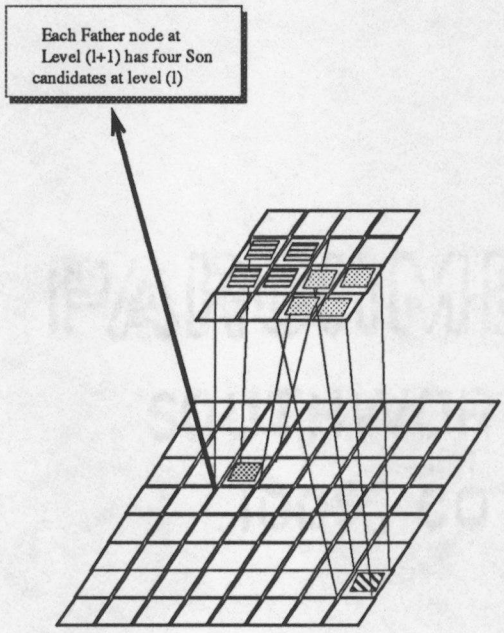



Figure 3.3: Relation of nodes in pyramid.

As shown in Figure 3.3, each node $I_l(i,j)$ has a 2 by 2 array of *candidate father* nodes at $I_{l+1}(i'',j'')$, where $i''$ and $j''$ are given by

$$i'' = i/2, i/2+1$$

$$j'' = j/2, j/2+1.$$

The base image is reflected by one row at the top and the bottom and one column at the left and the right edge (Figure 3.4).

### 3.1.2 Fuzzy Pyramid Linking

The theory of fuzzy sets was introduced by Zadeh in 1965 [44]. The theory of fuzzy sets is used to describe objects that cannot be grouped into classes that have sharply defined boundaries. The strength of relation between a son node and its candidate father node in pyramid linking is defined by the concept of fuzzy sets [37]. A son node is linked to the most similar candidate father node based on intensities. In non-fuzzy linking a son

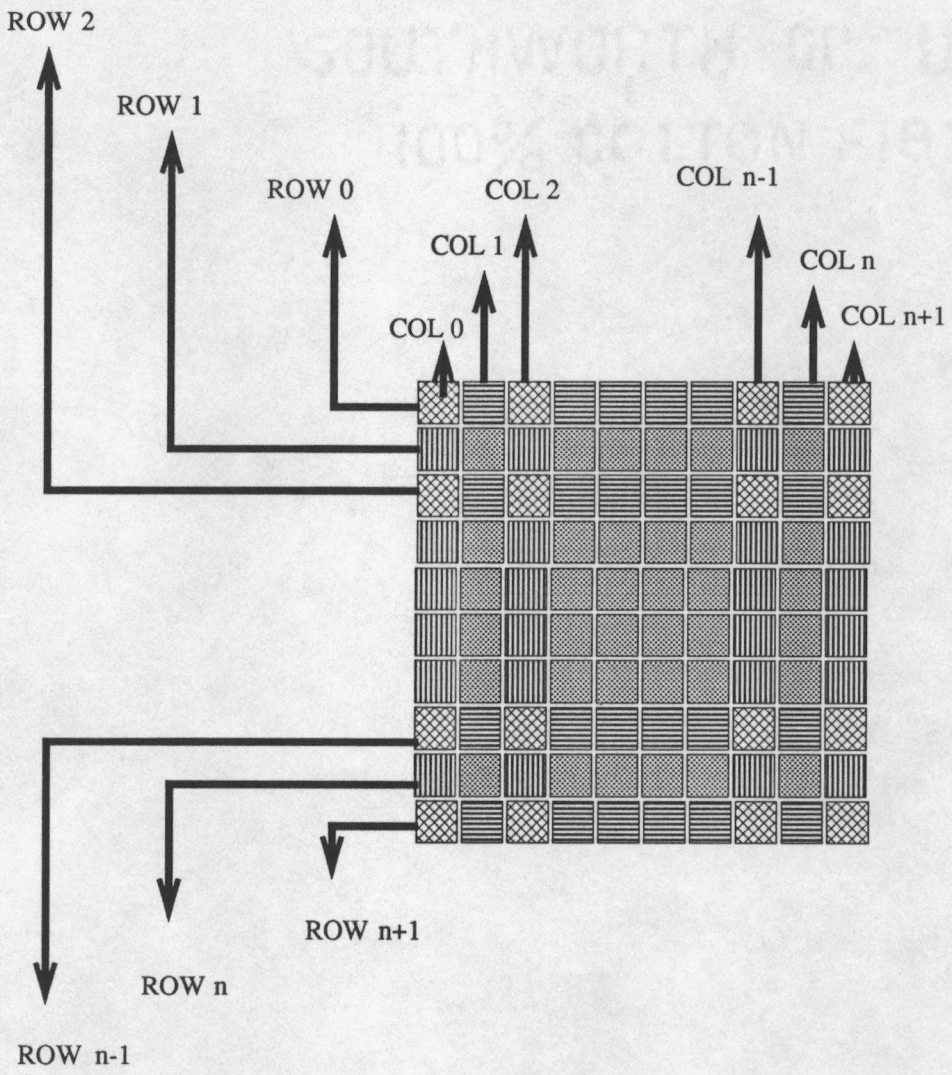



Figure 3.4: Image extension at each level of the pyramid.

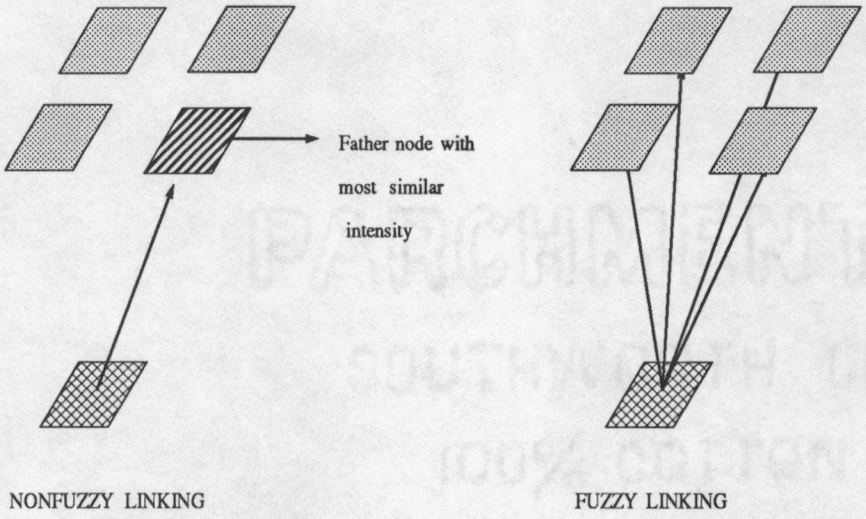



Figure 3.5: Types of links.

node is linked to the father node that has the greatest link strength (Figure 3.5). In fuzzy linking procedure the son node is linked to all four father candidates (Figure 3.5). The absolute difference in intensity between a son node and a candidate father node determines the link strength. Each link strength is associated with a membership grade. The membership grade is assigned based on membership function. The membership function defined as

$$LS(u;\alpha,\beta,\gamma) = 1 - S(u;\alpha,\beta,\gamma), \tag{3.1}$$

where

$$S(u;\alpha,\beta,\gamma) = \begin{cases} 1 & \text{for } u \le \alpha \\ 2\left(\frac{u-\alpha}{\gamma-\alpha}\right)^2 & \text{for } \beta \le u \le \gamma \\ 1 - 2\left(\frac{u-\gamma}{\gamma-\alpha}\right)^2 & \text{for } \alpha \le u \le \beta \\ 0 & \text{for } u \ge \gamma, \end{cases} \tag{3.2}$$

and $u = |I_l(i,j) - I_{l+1}(i'',j'')|$ is the absolute gray level difference between a son node and a candidate father node; $\alpha$ is the gray level difference below which the link strength

assigned is 1; $\gamma$ is the gray level difference above which the link strength assigned is 0; $\beta = \frac{\alpha+\gamma}{2}$ is the crossover point at which $LS(u;\alpha,\beta,\gamma) = 0.5$. The following variables are defined for iterative fuzzy linking:

- $t_l(i,j)$: The local image property is the weighted sum of the property values of the sons of the node. The image property at level zero is unchanged during the iteration process.
- $p_l(i,j)$: A pointer to the node's father in the level above which has the largest LS link strength value.
- $s_l(i,j)$: the link strength of the son nodes.

The iterations proceed in the following manner:

1. For level $l = 0$ set

$$s_0(i,j) = 1 \quad \text{and} \quad t_0(i,j) = I_0(i,j)$$

2. For each level $l$ from 1 to $N$ set

$$s_l(i,j) = \sum_{i'j'} s_{l-1}(i',j')LS(u;\alpha,\beta,\gamma)$$

and

$$t_l(i,j) = \sum_{i'j'} t_{l-1}(i',j')LS(u;\alpha,\beta,\gamma),$$

where $LS(u;\alpha,\beta,\gamma)$ is the strength of the link between node $(i,j)$ at level $l$ and its son $(i',j')$ at level $l-1$, and the summation is performed for all sons of the node.

3. For each node at level $l$, for $0 \leq l < N$, the pointer $p_l(i,j)$ points to the father node at level $l+1$ that has the largest LS (link strength) value among the four

candidate father nodes. If two or more fathers have the same LS value, then a link is chosen randomly. However, if the tied links include the link that existed in the previous iteration, the link remains unchanged.

4. Once the links have propagated to the top of the pyramid, every node, except those at level 0, is recomputed in the following manner:

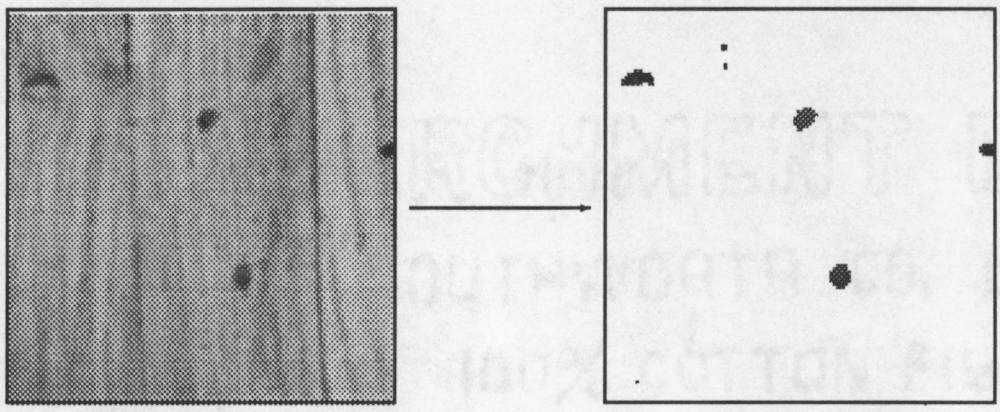
$$I_l(i,j) = t_l(i,j)/s_l(i,j) \quad for\ s_l(i,j) > 0.$$

5. If no link is reassigned during the current iteration, then a steady state is assumed. If any number of links has been reassigned during the current iteration, then the procedure is repeated starting from step 2.

6. Segmentation is achieved in one top-down pass starting from level $N-1$. In this pass, a son node at level $l$ is replaced by the father node that has the largest LS value, provided that the LS value is above the threshold strength value specified by the user. This threshold strength value, denoted by $\tau$, is selected based on the amount of detail required within the segmented image. A result of the image segmentation process with $\alpha = 5.0$ and $\tau = 0.5$ is shown in Figure 3.6.

## 3.2 Pyramid Linking in Distributed Memory Systems

In this section before discussing implementation of the algorithm, the flexibility of hypercubes is discussed and an efficient way of using the architecture for the purpose is shown. Different topologies can be implemented on the hypercube, such as tree structure, ring, and mesh. For image processing purposes, one dimensional or two dimensional mesh topology is best suited because images are stored in matrix form arrays. Depending on



Original image $I_0(i,j)$ Result image

Figure 3.6: Pyramid segmentation on a 128 by 128 image.

the size of the image array, one dimensional or two dimensional mesh can be used. One dimension mesh organization is easier to maintain than two dimensions. Two dimensional mesh organization is used only when the image array size is very large. The following sections describe how mesh organization is implemented on hypercube architecture, and then some algorithms which are used for ease of communicating between processors on hypercube are discussed. Finally, the pyramid segmentation algorithm implemented on the hypercube is discussed. The design of a parallel algorithm is divided into two parts: host processor and node processors. The host processor algorithm performs I/O operations and controls the node processors. The node processors cannot perform any type of operations except through the host processor by sending and receiving messages, as shown in Figure 3.7.

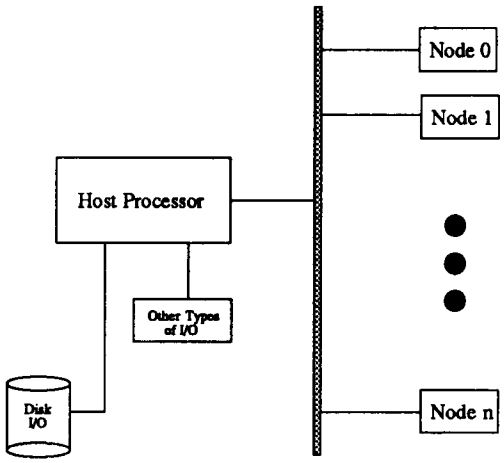



Figure 3.7: Host and node processors configuration.

### 3.2.1 Algorithm 1 (Creating Mesh on Hypercube)

Because of the flexibility of hypercube architecture, a mesh with $q$ dimensions can be embedded in a hypercube. The following is an algorithm to embed a $q$ dimensional mesh into a $k$ dimensional hypercube. Given a processor label from the hypercube, this algorithm will return the labels of the adjacent processors and locate the processor on the mesh. The information needed includes the processor label, the dimension of the mesh, and the number of processors in each direction of the mesh. This algorithm is as follows:

1. Round up the number of processors to the next power of two. If the total number of processors needed is larger than the available processors, then stop the process:

```
total = 1
for i = 1 to mesh-dimension do
    size(i) = 1
    while(size(i) < dimension) shift left size(i)
    total = total * size(i)
end for
if (total > total_cpus) error
```

2. Find the coordinates of the processor for each direction:

```
displacement = label
for i = 1 to dim do
    displacement = mod(displacement,size(i))
    coordinate(i) = invgray(displacement)
    displacement = displacement / size(i)
end for
```

3. Take out nodes outside the mesh:

```
for i = 1 to dim do
    if ( displacement > 0 or coordinate(i) >= dimension(i)) then
      for j = 1 to dim do
          coordinate(i) = -1
          predecessor(i) = -1
          successor(i) = -1
      end for
    end if
end for
```

4. Set predecessor and successors:

```
slice = 1
for i = 1 to dim do
    difference = gray(coordinate(i)-1) - gray(coordinate(i))
    predecessor(i) = label + difference*slice
    difference = gray(coordinate(i)) - gray(coordinate(i)+1)
    successor(i) = label - difference*slice
    slice = slice*size(i)
end for
```

5. Set boundary conditions (check for wraparound)

```
slice = 1
for i = 1 to dim do
    if (no wrap required) then
      if (coordinate(i) = 0) predecessor(i) = -1
      if (coordinate(i) = dimension(i)) successor(i) = -1
    else
      if (coordinate(i) = 0) predecessor(i) = label + gray(dimension(i))*slice
      if (coordinate(i) = dimension(i)) successor(i) = label - gray(dimension(i))*slice
    end if
    slice = slice * size(i)
end for
```

Figure 3.8 shows a one dimensional no-wrap mesh of size 8 embedded in a hypercube of size 3.

### 3.2.2 Algorithm 2 (Search for Global Minimum on Hypercube)

Given the number of processors on the hypercube, this algorithm finds the minimum value between processors. Processor $p_0$ is chosen to be the root processor which acts as the *Cube Manager*. Other processors communicate through the root processor. Figure 3.9 shows the steps involved, and the steps of finding global minimum follow:

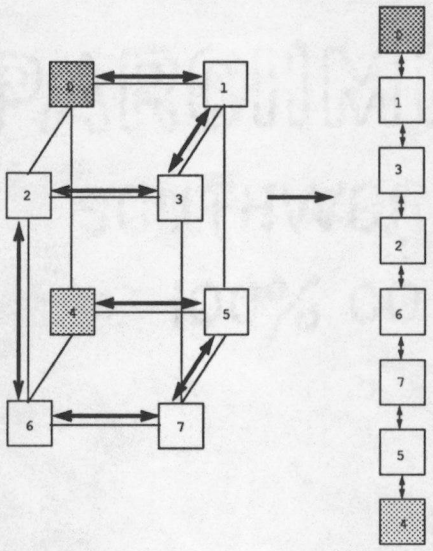



Figure 3.8: Mesh embedded in hypercube.

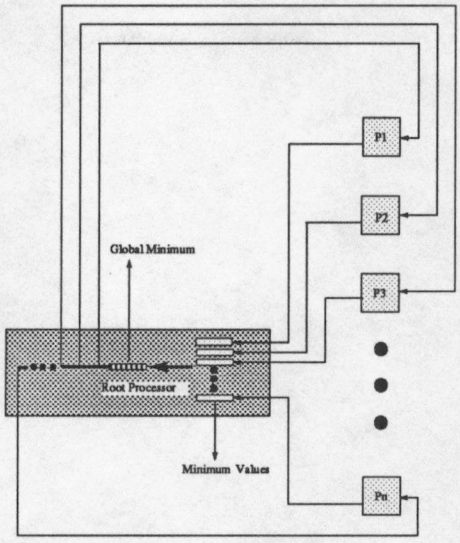



Figure 3.9: Search for global minimum value on hypercube.

if (processor = $P_{root}$) then

  for $p$ = 1 to num_procs - 1

    receive $min_p$ from $p$

    if ($min_{root} < min_P$) $min_{root} = min_P$

  end for

  broadcast $min_{root}$

else

  send $min_P$ to $P_{root}$

end if

This procedure is used to find the average of the base image of the pyramid. Section 3.2.3 shows a message broadcast from the *root processor* to other processors. The global maximum is also found through the same type of procedure.

### 3.2.3 Algorithm 3 (Broadcasting a Message on Hypercube)

The following algorithm shows a message broadcasted from the *root* processor through a *spanning tree.* Figure 3.10 shows an example for a hypercube of size 3. Maximum efficiency is obtained by minimizing the number of *hops* needed for communication routing:

1. Find the parent label of the processor on the tree.

   $P_{parent}$ = logical or ($P$ and $P_{root}$)

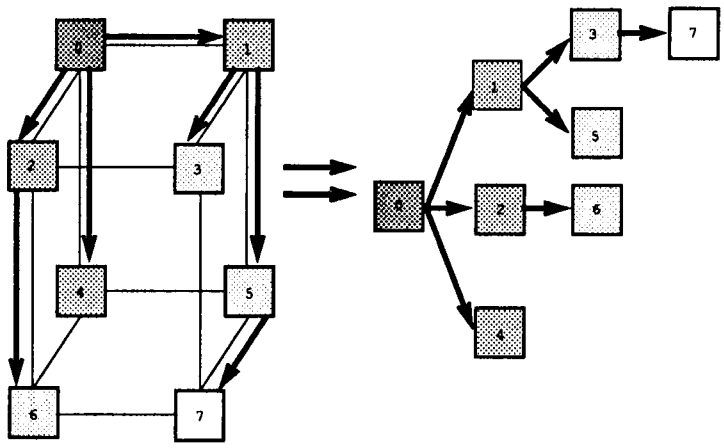



Figure 3.10: Broadcasting a message in hypercube.

2. Receive a message from the parent and send it to the children processors.

if ($P \neq P_{root}$) receive message from $P_{parent}$

$P_{child} = 1$

while ($P_{child}$ < n_processors) do

    if ($P < P_{child}$) send message to $P_{child}$

end do

This procedure is used to share information between processors at each iteration of pyramid linking.

### 3.2.4 Embedding the Pyramid

The base image is a dimension of $2^N$ by $2^N$ and is divided by the number of of processors $P$. Each sub-image is extended at the top and bottom by one row; therefore, the size of each sub-image $I_0^p$ is $\frac{2^N}{P} + 2$ by $2^N + 2$, where $P$ is the number of processors and $p$ is the processor label of the sub-image. The sub-image reflection at the edges shown in Figure 3.11 is performed as follows:

1. if processor $p$ has predecessor processor $p-1$, then

   - send row 2 of $I_0^p$ to processor $p-1$
   - receive row 1 of $i_0^p$ from processor $p-1$

2. if processor $p$ does not have predecessor processor, then

   - extend row 1 by copying row 2

3. if processor $p$ has successor processor $p+1$, then

   - send row $\frac{2^N}{P}+1$ of $I_0^p$ to processor $p+1$
   - receive row $\frac{2^N}{P}+2$ of $i_0^p$ from processor $p+1$

4. if processor $p$ does not have successor processor, then

   - extend to row $\frac{2^N}{P}+2$ by coying row $\frac{2^N}{p}$ 2

Figure 3.11 shows an example of a 128 by 128 pixel image that is divided into 4 sub-images. The algorithm described in Section 3.2.1 is used to obtain information about each processor coordination and the processor's neighbors in a mesh organization. The row interchange of sub-images at each processor must be done before construction at the next level of the image pyramid. Each node or pixel of the pyramid level $l$ is denoted by $I_l^p(i,j)$, where $p$ is the processor tag, $i$ is the row, and $j$ is the column of the sub-image in the pyramid. Because the size of the sub-images is reduced by 2 at a level $L$ of the size of the sub-image $I_L^p$ is 1 by $P$, the pyramid cannot fit into the mesh organization beyond this level; therefore, each sub-image $I_L^p$ is sent to processor $p_0$, and processor $p_0$ constructs the levels after level $L$. If the base level of pyramid $I_0$ cannot be divided into the number of processors the algorithm is performed using the *root* processor. Each

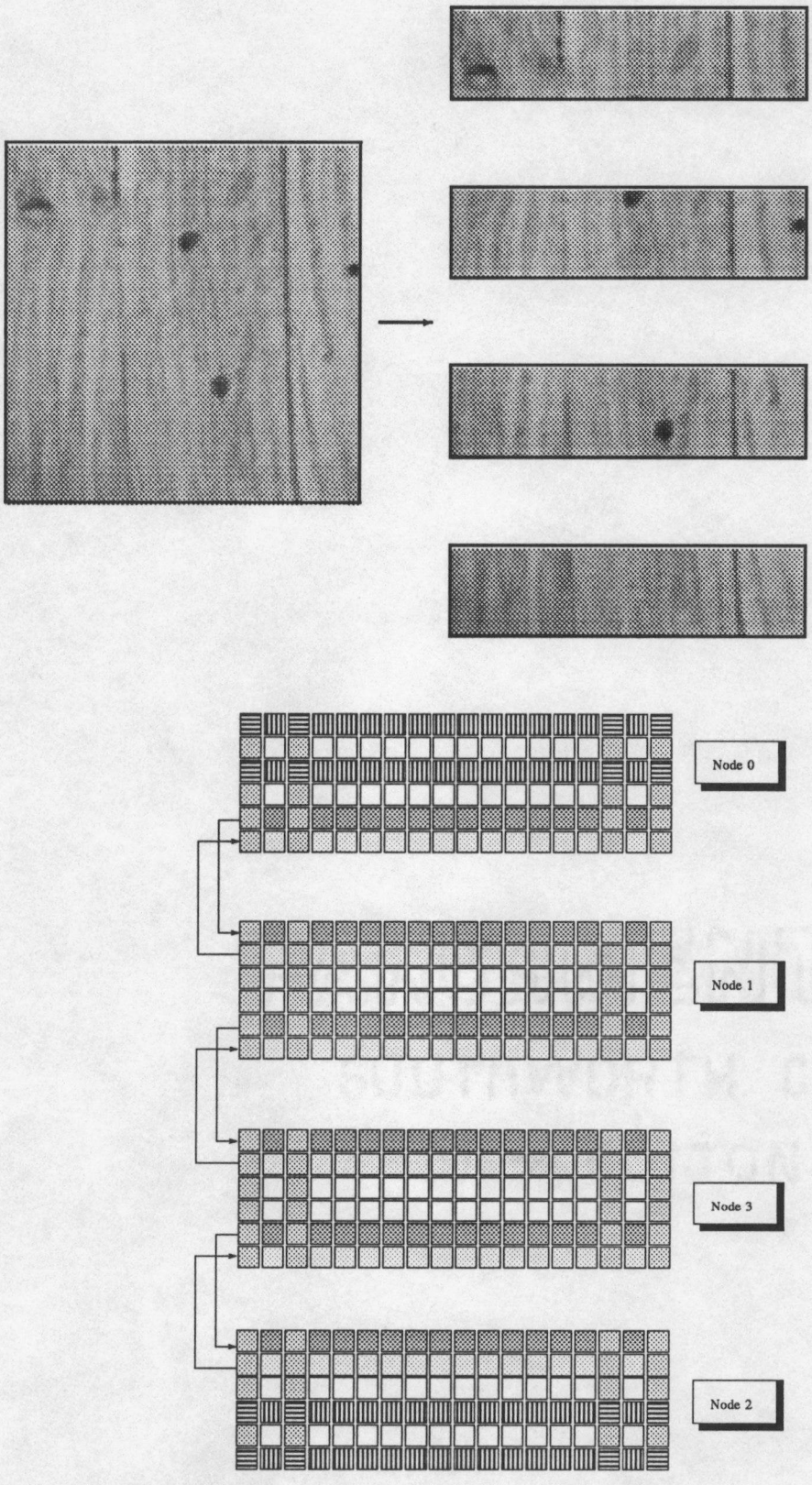



Figure 3.11: Base level of pyramid divided into 4 processors.

node $I_l^p(i,j)$ is constructed by taking the weighted average of $I_0^p$ nodes within a 4 by 4 window as given by

$$I_l^p(i,j) = \sum_{m=-2}^{1} \sum_{n=-2}^{1} w(m,n) I_{l-1}^p(2i+m, 2j+n) \quad for \;\; 0 \leq i,j < N_l^p, \; l <= L, \; and \; p \neq 0,$$

where $N_l^p$ represents the size of the sub-image in processor $p$, i.e., $\frac{N_l X N_l}{P}$($P$ represents the number of processors, and p is the processor tag). The generation of kernel $w$ is discussed in Section 3.1.1. Each node at level $l$, for $l > 0$, has a 4 by 4 array of *candidate son* nodes at level $l-1$, and each node at level $l$ below the top level of the pyramid has a 2 by 2 array of *candidate father* nodes. Because the size of the sub-images is reduced by a factor of two, at level $L$ the size of $I_L^p$ is 1 by $p$. This is actually a row of the entire image at level $L$. Level $L$ is the level at which the pyramid cannot be divided into the number of processors. At this level the processors cannot continue constructing the upper levels of the pyramid in parallel; therefore, the process should be done by one processor. All the processors except for processor zero send the sub-image at level $L$ to processor zero, and processor zero builds $I_L^{p_0}$, which is same as $I_L$. Processor zero constructs levels $l > L$ as follows:

$$I_l^{p_0}(i,j) = \sum_{m=-2}^{1} \sum_{n=-2}^{1} w(m,n) I_{l-1}^{p_0}(2i+m, 2j+n) \quad for \;\; 0 \leq i,j < N_l, \; l > L, \; and \; p = 0$$

### 3.2.5 Fuzzy Pyramid Linking on Distributed Memory Systems

The steps performed in the Fuzzy Pyramid Linking algorithm can be divided among several processors by using message passing strategies. At steps of the parallel algorithm, processors need information from adjacent processors or global information from other processors. This is needed to obtain the correct information from the neighborhood of the pixel or global information of the image such as maximum or minimum pixel value. These types of information are obtained by passing messages between the processors,

using the techniques described in previous sections in this chapter. At level $L$ of the pyramid the number of rows of $I_L$cannot be divided to the processors; then segmentation process on levels higher than level $L$ is performed by one processor. The result is then propagated to the other processors as needed. The mesh algorithm is used to embed the pyramid into the processors. The following variables are defined for the linking and ensuing iterative process in each processor in a distributed memory system:

- $t_l^p(i,j)$: The image property is the weighted sum of the property values of the sons of the node at processor $p$. Because the son candidates are in a 4 by 4 window, there are overlaps of son candidates. Therefore, $t_l^p$ is extended by one row at the top from predecessor processor $t_l^{p-1}$ if $p-1$ exists, and by one row at the bottom from $t_l^{p+1}$ if processor $p+1$ exists. This applies to all levels $l < L$. For levels $l \geq L$ only $t_l^0$ exists and is the same as $t_l$.

- $p_l^p(i,j)$: The pointer to the node's father in the level above has the largest LS link strength value.

- $s_l^p(i,j)$: The strength value is the strength of weight values of the son nodes. Since the son candidates are in a 4 by 4 window, there would be overlaps of son candidates. Therefore, $s_l^p$ is extended by one row at the top from $s_l^{p-1}$ if $p-1$ exists, and by one row at bottom from $s_l^{p+1}$ if processor $p+1$ exists. This applies to all levels $l < L$. For levels $l \geq L$, only $s_l^0$ exists and is the same as $s_l$.

The iterations proceed in the following manner:

1. For level $l = 0$ set

$$s_0^p(i,j) = 1 \quad \text{and} \quad t_0^p(i,j) = I_0^p(i,j).$$

2. Perform row interchange on $I$ between $p$ and the adjacent processors:

- If $p_m$ has a predecessor processor $p_{m-1}$ (Figure 3.12a), then for each level $l$ from 1 to $L$
  - send $I_l^{p_m}(2,j)$ to $I_l^{p_{m-1}}(last,j)$
  - receive $I_l^{p_m}(1,j)$ from $I_l^{p_{m-1}}(last-1,j)$
- If $p_m$ has a successor $p_{m+1}$ (Figure 3.12a), then for each level $l$ from 1 to $L$
  - send $I_l^{p_m}(last-1,j)$ to $I_l^{p_{m+1}}(1,j)$
  - receive $I_l^{p_m}(last,j)$ from $I_l^{p_{m+1}}(2,j)$

  where $last$ is the last row of $I_l^p$.

To accelerate the process of communication, the messages to be sent between the processors are buffered. This way, instead of sending messages for each level $l$, which would be $l$ messages, in one message all levels are packed and unpacked.

3. For each level $l$ from 1 to $L$ set

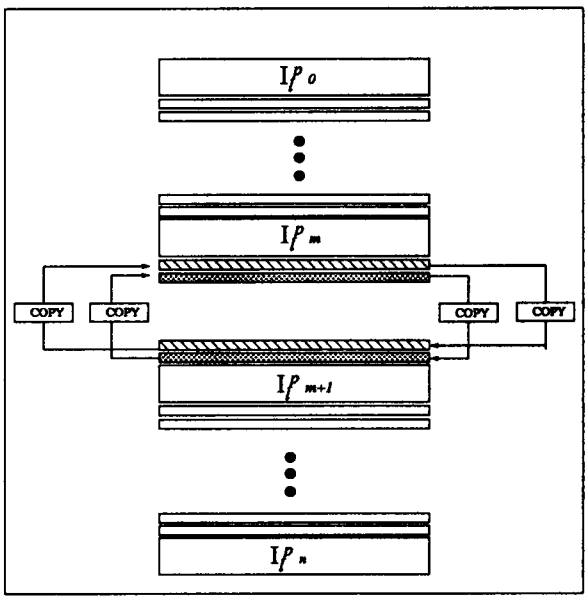
$$s_l^p(i,j) = \sum_{i'j'} s_{l-1}^p(i',j')LS(u;\alpha,\beta,\gamma)$$

and

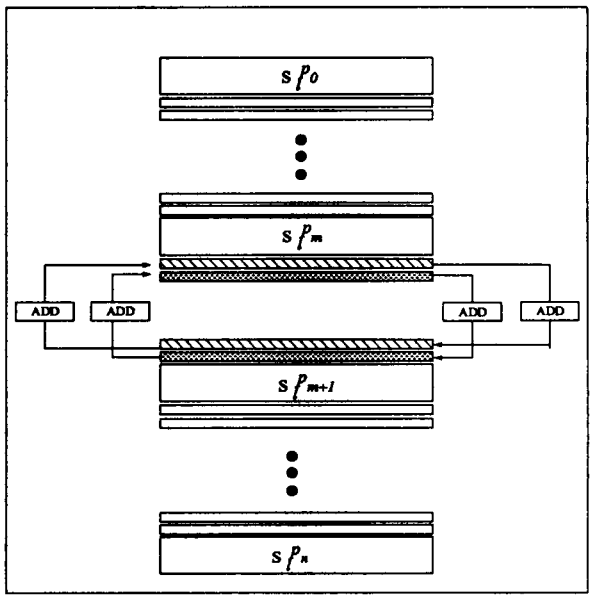
$$t_l^p(i,j) = \sum_{i'j'} t_{l-1}^p(i',j')LS(u;\alpha,\beta,\gamma)$$

The link strength $LS(u;\alpha,\beta,\gamma)$ is explained in Section 3.1.2. Perform row interchange for $t_l^p$, as shown in Figure 3.12a, and addition on $t_l^p$ and $s_l^p$, as shown in Figure 3.12b and 3.12c, between the adjacent processors:

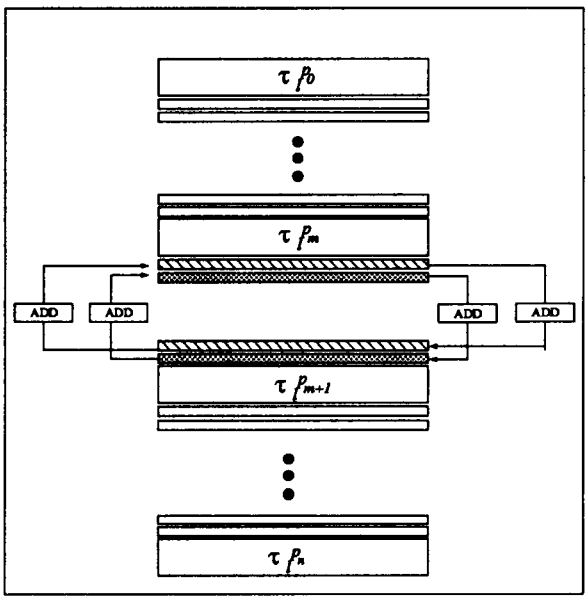
- If $p_m$ has a predecessor processor $p_{m-1}$, then




a. Subimages




b. Sum of strength values




c. Sum of weighted intensity values

Figure 3.12: Row interchange between the neighbor processors.

- send $t_l^{p_m}(1,j)$ and add to $t_l^{p_{m-1}}(last-1,j)$
- receive $t_l^{p_{m-1}}(last-1,j)$ and add to $t_l^{p_m}(1,j)$
- send $s_l^{p_m}(1,j)$ and add to $s_l^{p_{m-1}}(last-1,j)$
- receive $s_l^{p_{m-1}}(last-1,j)$ and add to $s_l^{p_m}(1,j)$,

where $last$ is the last row of the $t_l^{p-1}$ and $s_l^{p-1}$.

- If $p$ has a successor processor $p_{m+1}$, then
  - send $t_l^{p_m}(last-1,j)$ and add to $t_l^{p_{m+1}}(1,j)$
  - receive $t_l^{p_{m+1}}(2,j)$ and add to $t_l^{p_m}(last-1,j)$
  - send $s_l^{p_m}(last-1,j)$ and add to $s_l^{p_{m+1}}(1,j)$
  - receive $s_l^{p_{m+1}}(2,j)$ and add to $s_l^{p_m}(last,j)$,

  where $last$ is the last row of the $t_l^p$ and $s_l^p$. All the message interchanging between processors is done by packing and unpacking messages to save communication time.

4. From all the processors $p \neq 0$ send $t_L^p$ and $s_L^p$ to processor $p_0$ and construct $t_L^{p_0}$ and $s_L^{p_0}$ (Figure 3.13).

5. For each level $l$ from $L$ to $N$ set

$$s_l^{p_0}(i,j) = \sum_{i'j'} s_{l-1}^{p_0}(i',j')LS(u;\alpha,\beta,\gamma)$$

and

$$t_l^{p_0}(i,j) = \sum_{i'j'} t_{l-1}^{p_0}(i',j')LS(u;\alpha,\beta,\gamma)$$

The link strength $LS(u;\alpha,\beta,\gamma)$ is explained in Section 3.1.1.

6. Once the links have propagated to the top of the pyramid, every node, except those at level 0, is recomputed in the following manner:

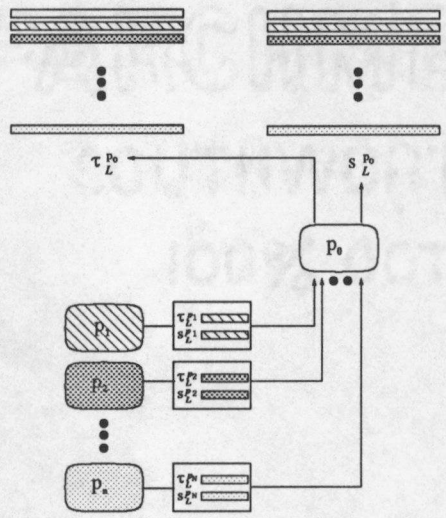


Figure 3.13: Construction of $t$ and $s$ in root processor.

- for processors $p \neq p_0$, for all levels $l$ from 1 to $L$ set

$$I_l^p(i,j) = t_l^p(i,j)/s_l^p(i,j) \quad for\ s_l^p(i,j) > 0$$

- for processors $p_0$, for all levels $l$ from 1 to $N$ set

$$I_l^{p_0}(i,j) = t_l^{p_0}(i,j)/s_l^{p_0}(i,j) \quad for\ s_l^{p_0}(i,j) > 0$$

7. At the end of each iteration every processor finds the number of reassigned links locally and communicates with processor $p_0$ to obtain the total number of reassigned links. If no link is reassigned during the current iteration, then a steady state is assumed. If any number of links has been reassigned during the current iteration, then the procedure is repeated starting from step 2.

Segmentation is achieved in one top-down pass starting from level $N-1$ down to level $L$ at processor $p_0$. The image $I_L^{p_0}$ is then propagated to other processors $p$ from $p_0$, and segmentation is achieved in a top-down pass starting from level $L$ down to level zero

in parallel as shown in Figure 3.14. In this pass, a son node at level $l$ is replaced by the father node that has the largest LS value, provided that the LS value is above the threshold strength value specified by the user. This threshold strength value, denoted by $\tau$, is selected based on the amount of detail required within the segmented image. Figure 3.15 shows the result of segmentation using 4 distributed memory processors. Appendix A represents a flow-chart of the algorithm.

## 3.3 Segmentation Using Shared Memory Systems

For shared memory systems there is no need for communication between the processors for determining global information because the global memory is available to all processors. This saves time that is spent for communication between processors as a result of passing information such as finding global minimum value between processors (discussed in Section 3.2.2) or broadcasting a message to all processors (described in Section 3.2.3), but there is still a need for synchronization between processors for shared memory systems. The implementation of pyramid segmentation for shared memory system is described in the subsequent sections. Images at each level of the pyramid are divided among the processors, but all reside in the global memory that is accessed by all the processors; therefore, there is no need for row interchange between processors as was needed for distributed memory systems.

### 3.3.1 Pyramid Creation in Shared Memory Systems

The pyramid base $I_0$, is $2^N$ by $2^N$ and resides in the global memory available to all $P$ processors. Each processor operates on part of $I_0$. The size of each of these sub-images $I_0^p$ is $\frac{2^N}{P}$ by $2^N$ except for the sub-image $I_0^{p_0}$, which extends by one row from the top,

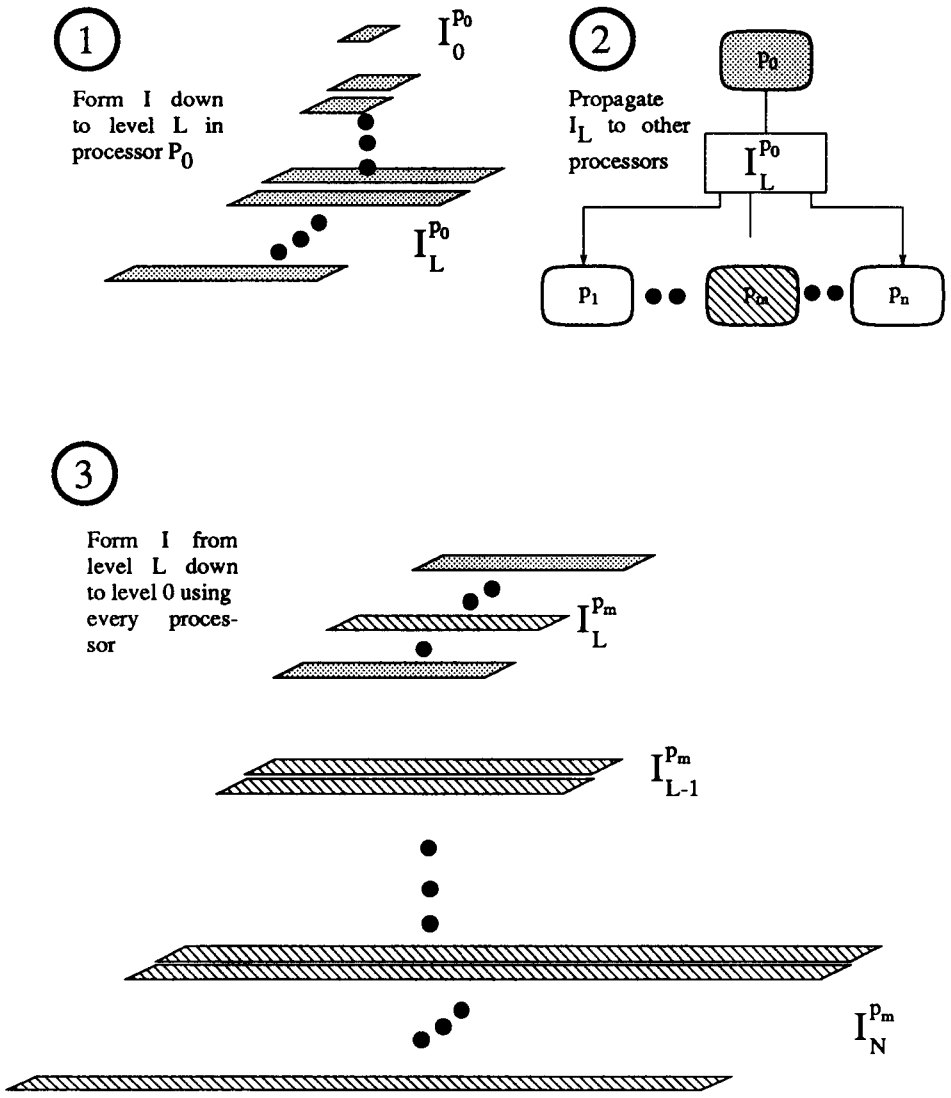



Figure 3.14: Steps of pyramid segmentation in distributed memory systems.

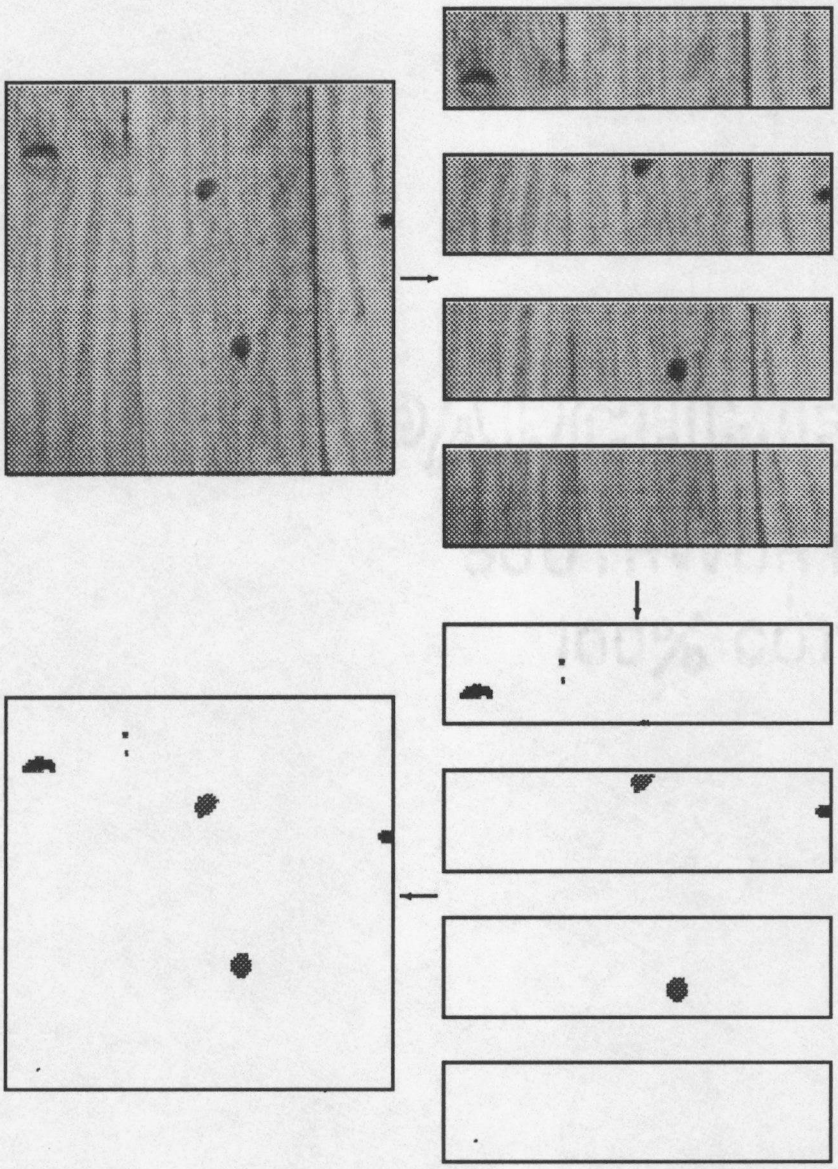


Figure 3.15: Pyramid segmentation result by 4 processors.

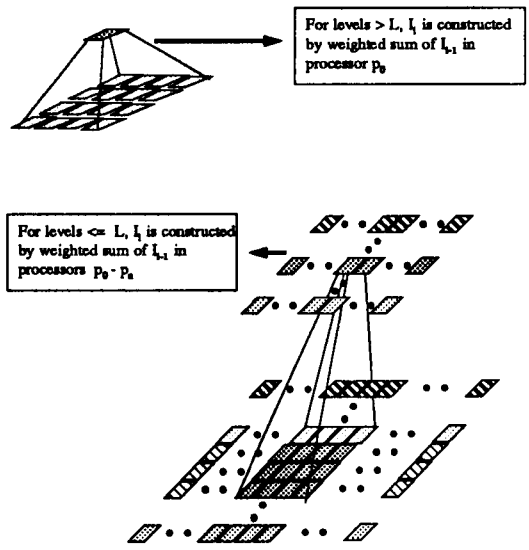



Figure 3.16: Pyramid construction in distributed memory systems.

and $I_0^{plast}$, which extends from the bottom by one row. Each node $I_l^p(i,j)$ is created by taking the weighted average of $I_{l-1}^p$ (Figure 3.16) given by

$$I_l^p(i,j) = \sum_{m=-2}^{1}\sum_{n=-2}^{1} w(m,n) I_{l-1}^p(2i+m, 2j+n) \; for \; 0 \le i,j < N^p l, \; l <= L, \; and \; p \neq 0,$$

where $N_l^p$ represents the size of the sub-image in processor $p$, i.e., $\frac{N_l X N_l}{P}$($P$ represents the number of processors). Processor $p_0$ generates the pyramid for levels greater than L as follows:

$$I_l^{p_0}(i,j) = \sum_{m=-2}^{1}\sum_{n=-2}^{1} w(m,n) I_{l-1}^{p_0}(2i+m, 2j+n) \; for \; 0 \le i,j < Nl, \; l > L, \; and \; p = 0.$$

The generation of kernel $w$ is discussed in Section 3.1.1. Since the window size of the weighted sum is 4 by 4, there are overlaps of the processor sub-images, as shown in Figure 3.16.

### 3.3.2 Fuzzy Pyramid Linking in Shared Memory Systems

In shared memory systems the implementation is done by dividing the task into sub-tasks. The number of sub-tasks is the number of processors available. The number of rows of images in the pyramid at each level can be divided into the number of rows for base level and levels $l \leq L$. Therefore, the fuzzy pyramid linking algorithm is performed on levels $l > L$, using the *root* processor $p_0$, while other processors wait. The following variables are defined for the linking and ensuing iterative process in each processor in shared memory systems:

- $t_l^p(i,j)$: The image property is the weighted sum of the property values of the sons of the node. The image property $t_l^p(i,j)$ is part of $t_l$, and since the window of son candidates is a 4 by 4 window, there are overlaps at rows in neighborhood processors. This applies to all levels $l \leq L$. For levels $l > L$, only $t_l^0$ exists and is the same as $t_l$.

- $s_l^p(i,j)$: The strength value is the strength of the weight values of the sons of the node. Each of these sub-images is part of $s_l$, and since the window of the son candidates is a 4 by 4 window, there are overlaps at rows in neighborhood processors. This applies to all levels $l <= L$. For level $l > L$, only $s_l^0$ exists and is the same as $s_l$.

- $p_l^p(i,j)$: The pointer to the node's father in level $l+1$, which has the largest link strength value LS.

The iterations proceed in the following manner:

1. For level $l = 0$ set

$$s_0^p(i,j) = 1 \quad \text{and} \quad t_0^p(i,j) = I_0^p(i,j)$$

Perform synchronization.

2. For each level $l$ from 1 to $L$ set

$$s_l^p(i,j) = \sum_{i'j'} s_{l-1}^p(i',j')LS(u;\alpha,\beta,\gamma) t_l^p(i,j) = \sum_{i'j'} t_{l-1}^p(i',j')LS(u;\alpha,\beta,\gamma)$$

Perform synchronization.

3. For each level $l$ from $L$ to $N$ set

$$s_l^{p_0}(i,j) = \sum_{i'j'} s_{l-1}^{p_0}(i',j')LS(u;\alpha,\beta,\gamma)$$

and

$$t_l^{p_0}(i,j) = \sum_{i'j'} t_{l-1}^{p_0}(i',j')LS(u;\alpha,\beta,\gamma)$$

$LS(u;\alpha,\beta,\gamma)$ is explained in Section 3.1.2. While this step is done in processor $p_0$, other processors wait until processor $p_0$ is done.

4. Once the links have propagated to the top of the pyramid, every node, except those at level 0, is recomputed in the following manner:

   - for processors $p \neq p_0$, for all levels $l$ from 1 to $L$ set

$$I_l^p(i,j) = t_l^p(i,j)/s_l^p(i,j) \quad for\ s_l^p(i,j) > 0$$

   - for processor $p_0$, for all levels $l$ from 1 to $N$ set

$$I_l^{p_0}(i,j) = t_l^{p_0}(i,j)/s_l^{p_0}(i,j) \quad for\ s_l^{p_0}(i,j) > 0$$

   - Perform synchronization.

5. At the end of each iteration every processor finds the number of reassigned links, and then the total number of reassigned links is computed. If no link is reassigned during the current iteration, then a steady state is assumed. If any number of links has been reassigned during the current iteration, then the procedure is repeated starting from step 2.

Segmentation is achieved in one top-down pass starting from level $N-1$ down to level $L$ at processor $p_0$, while other processors wait. Then segmentation is achieved starting from level $L$ down to level 0 using all processors, as shown Figure 3.17. In this top down pass, a son node at level $l$ is replaced by the father node that has the largest LS value, provided that the LS value is above the threshold strength value specified by the user. This threshold strength value, denoted by $\tau$, is selected based on the amount of detail required within the segmented image. Figure 3.17 illustrates the steps of parallel pyramid segmentation using 4 processors in a shared memory system. Appendix B presents a flow-chart of the algorithm.

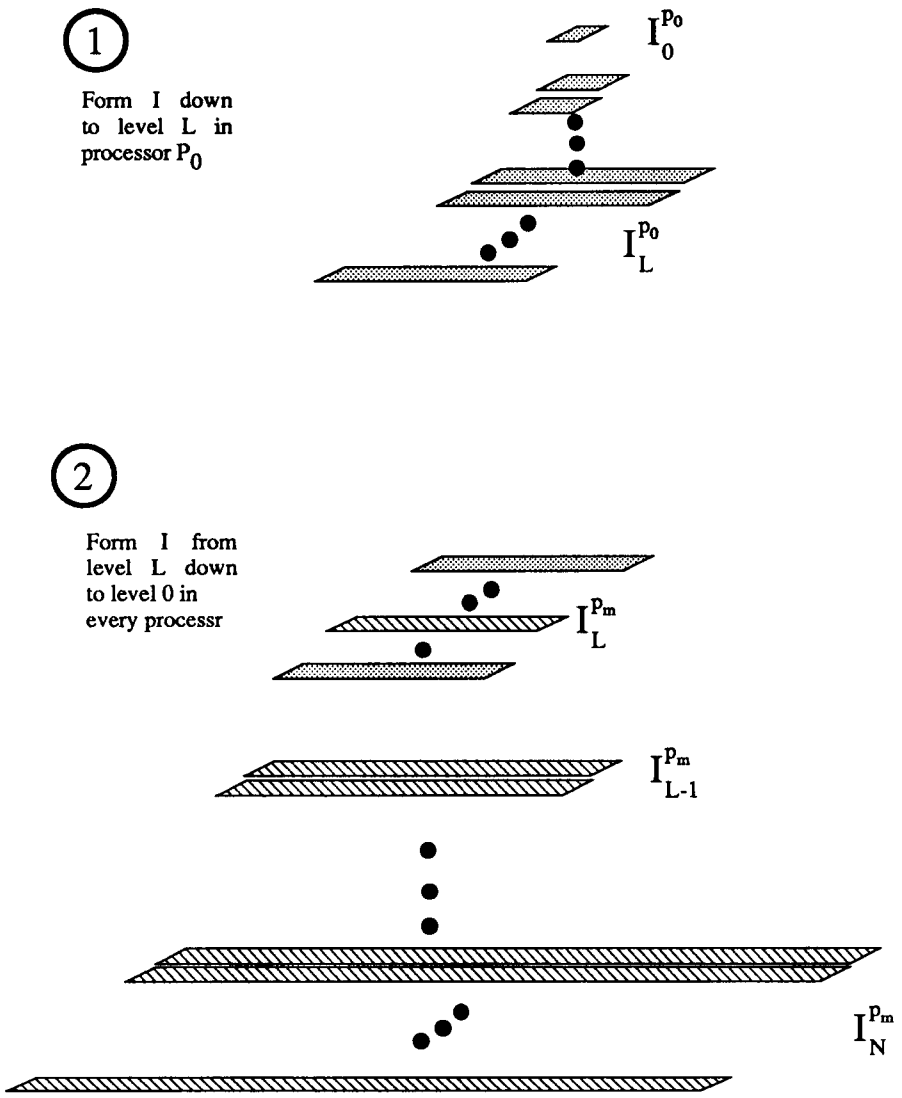



Figure 3.17: Steps of pyramid segmentation in shared memory systems.

# CHAPTER 4

## EXPERIMENTAL RESULTS

The performance of fuzzy pyramid linking is analyzed and presented for single processor, shared memory, and distributed memory systems in this chapter. The algorithms are reduced to the following steps:

1. Calculate the mean of base image $I_0$.
2. Create and initialize the pyramid.
3. Iterate fuzzy link.
4. Perform segmentation.
5. Scale the resulting image.

For the analysis the symbols have the following meanings:

1. $\boxed{+}$ Addition.
2. $\boxed{-}$ Subtraction.
3. $\boxed{\times}$ Multiplication.
4. $\boxed{\div}$ Division.
5. $\boxed{\text{Cond}}$ Conditional statements.
6. $\boxed{\text{Sync}}$ Synchronization for shared memory systems.

7. [Send] Send messages for distributed memory systems.

8. [Receive] Receive messages for distributed memory systems.

The number of operations required for each of the algorithms for each step is as follows:

1. Calculate the mean of the base image.

   (a) Single processor:

$$OP = (2^N \times 2^N)[+] + 1[\div]$$

   (b) Shared memory system with $N_p$ processors:

$$OP_p = \frac{(2^N \times 2^N)}{N_P}[+] + 1[\div] + 1[\text{Sync}] \quad for \quad p \neq 0$$

*and*

$$OP_{p_0} = \frac{(2^N \times 2^N)}{N_p}[+] + 2[\div] + 1[\text{Sync}]$$

   (c) Distributed memory systems with $N_P$ processors:

$$OP_p = \frac{(2^N \times 2^N)}{N_P}[+] + 1[\div] + 1[\text{Send}] + 1[\text{Receive}] \quad for \quad P \neq 0$$

*and*

$$OP_{p_0} = \frac{(2^N \times 2^N)}{N_p}[+] + 2[\div] + (N_p - 1)[\text{Send}] + (N_p - 1)[\text{Receive}]$$

2. Generate and initialize the pyramid.

   To generate:

   (a) Single processor:

$$OP \approx \sum_{l=0}^{N-1} \frac{(16 \times 2^{2(N-l)})}{(l+1)^2}[\times] + \sum_{l=0}^{N-1} \frac{(2^{2(N-l)})}{(l+1)^2}[+] + \sum_{l=1}^{N} \frac{(2^{2(N-l)})}{(l+1)^2}[=]$$

(b) Shared memory system with $N_p$ of processors:

$$OP_p \approx \sum_{l=0}^{L} \frac{(16\times 2^{2(N-l)})}{N_p\times(l+1)^2}\boxed{\times} + \sum_{l=0}^{L} \frac{(2^{2(N-l)})}{N_p\times(l+1)^2}\boxed{+} + \sum_{l=1}^{L+1} \frac{(2^{2(N-l)})}{N_p\times(l+1)^2}\boxed{=} + (L+2)\boxed{\text{Sync}} \quad for \quad p \neq 0$$

*and*

$$OP_{p_0} \approx \left\{\sum_{l=0}^{L} \frac{(16\times 2^{2(N-l)})}{N_p\times(l+1)^2} + \sum_{l=L+1}^{N-1} \frac{(16\times 2^{2(N-l)})}{(l+1)^2}\right\}\boxed{\times} + \left\{\sum_{l=0}^{L} \frac{(2^{2(N-l)})}{N_p\times(l+1)^2} + \sum_{l=L+1}^{N-1} \frac{(2^{2(N-l)})}{(l+1)^2}\right\}\boxed{+} + \left\{\sum_{l=1}^{L=1+1} \frac{(2^{2(N-l)})}{N_p\times(l+1)^2} + + \sum_{l=L+2}^{N} \frac{(2^{2(N-l)})}{(l+1)^2}\right\}\boxed{=} + (L+2)\boxed{\text{Sync}}$$

(c) Distributed memory system with $N_p$ processors:

$$OP_p \approx \sum_{l=0}^{L} \frac{(16\times 2^{2(N-l)})}{N_p\times(l+1)^2}\boxed{\times} + \sum_{l=0}^{L} \frac{(2^{2(N-l)})}{N_p\times(l+1)^2}\boxed{+} + \sum_{l=1}^{L+1} \left[\frac{(2^{2(N-l)})}{N_p\times(l+1)^2} + \frac{(4\times 2^{N-l})}{(l+1)^2}\right]\boxed{=} + (2L+1)\boxed{\text{Send}} + (2L)\boxed{\text{Receive}} \quad for \quad p \neq 0$$

$$OP_{p_0} \approx \left\{\sum_{l=0}^{L} \frac{(16\times 2^{2(N-l)})}{N_p\times(l+1)^2} + \sum_{l=L+1}^{N-1} \frac{(16\times 2^{2(N-l)})}{(l+1)^2}\right\}\boxed{\times} + \left\{\sum_{l=0}^{L} \frac{(2^{2(N-l)})}{N_p\times(l+1)^2} + \sum_{l=L+1}^{N-1} \frac{(2^{2(N-l)})}{(l+1)^2}\right\}\boxed{+} + \left\{\sum_{l=1}^{L=1+1}\left[\frac{(2^{2(N-l)})}{N_p\times(l+1)^2} + \frac{(4\times 2^{(N-l)})}{(l+1)^2}\right] + \sum_{l=L+2}^{N} \frac{(2^{2(N-l)})}{(l+1)^2}\right\}\boxed{=} + (L)\boxed{\text{Send}} + (L+N_p-1)\boxed{\text{Receive}}$$

To initialize:

(a) Single processor:

$$OP = \sum_{l=0}^{N} \frac{(4\times 2^{2(N-l)})}{(l+1)^2}\boxed{=}$$

(b) Shared memory system with $N_P$ processors:

$$OP_p = \sum_{l=0}^{L} \frac{(4\times 2^{2(N-l)})}{N_p\times(l+1)^2}\boxed{=} + 1\boxed{\text{Sync}} \quad for \quad p \neq 0$$

*and*

$$OP_{p0} = \sum_{l=0}^{L} \frac{(4\times 2^{2(N-l)})}{N_p\times(l+1)^2} + \sum_{l=L+1}^{N} \frac{(4\times 2^{2(N-l)})}{(l+1)^2}\boxed{=} + 1\boxed{\text{Sync}}$$

(c) Distributed memory system with $N_P$ processors:

$$OP_p = \sum_{l=0}^{L} \frac{(4\times 2^{2(N-l)})}{N_p\times(l+1)^2}\boxed{=} \quad for \;\; p \neq 0$$

*and*

$$OP_{p0} = \sum_{l=0}^{L} \frac{(4\times 2^{2(N-l)})}{N_p\times(l+1)^2} + \sum_{l=L+1}^{N} \frac{(4\times 2^{2(N-l)})}{(l+1)^2}\boxed{=}$$

3. Iterate fuzzy link.

Generate links:

(a) Single processor:

$$OP \approx \sum_{l=0}^{N} \frac{(60\times 2^{2(N-l)})}{(l+1)^2}\boxed{=} + \sum_{l=0}^{N} \frac{(6\times 2^{2(N-l)})}{(l+1)^2}\boxed{+} + \sum_{l=0}^{N} \frac{(18\times 2^{2(N-l)})}{(l+1)^2}\boxed{-} +$$
$$\sum_{l=0}^{N} \frac{(8\times 2^{2(N-l)})}{(l+1)^2}\boxed{\times} + \sum_{l=0}^{N} \frac{(2^{2(N-l)})}{(l+1)^2}\boxed{\div} + \sum_{l=0}^{N} \frac{(25\times 2^{2(N-l)})}{(l+1)^2}\boxed{\text{Cond}}$$

(b) Shared memory system with $N_P$ processors:

$$OP_p \approx \sum_{l=0}^{L} \frac{(60\times 2^{2(N-l)})}{N_p\times(l+1)^2}\boxed{=} + \sum_{l=0}^{L} \frac{(6\times 2^{2(N-l)})}{N_p\times(l+1)^2}\boxed{+} +$$
$$\sum_{l=0}^{L} \frac{(18\times 2^{2(N-l)})}{N_p\times(l+1)^2}\boxed{-} + \sum_{l=0}^{L} \frac{(8\times 2^{2(N-l)})}{N_p\times(l+1)^2}\boxed{\times} +$$
$$\sum_{l=0}^{L} \frac{(2^{2(N-l)})}{N_p\times(l+1)^2}\boxed{\div} + \sum_{l=0}^{L} \frac{(25\times 2^{2(N-l)})}{N_p\times(l+1)^2}\boxed{\text{Cond}} +$$
$$(2L+1)\boxed{\text{Lock}} + (L+2)\boxed{\text{Sync}} \quad for \;\; p \neq 0$$

*and*

$$OP_{p0} \approx \left\{\sum_{l=0}^{L} \frac{(60\times 2^{2(N-l)})}{N_p\times(l+1)^2} + \sum_{l=L+1}^{N} \frac{(60\times 2^{2(N-l)})}{N_p\times(l+1)^2}\right\}\boxed{=} +$$
$$\left\{\sum_{l=0}^{L} \frac{(6\times 2^{2(N-l)})}{N_p\times(l+1)^2} + \sum_{l=L+1}^{N} \frac{(6\times 2^{2(N-l)})}{N_p\times(l+1)^2}\right\}\boxed{+} +$$
$$\left\{\sum_{l=0}^{L} \frac{(18\times 2^{2(N-l)})}{N_p\times(l+1)^2} + \sum_{l=L+1}^{N} \frac{(18\times 2^{2(N-l)})}{N_p\times(l+1)^2}\right\}\boxed{-} +$$
$$\left\{\sum_{l=0}^{L} \frac{(8\times 2^{2(N-l)})}{N_p\times(l+1)^2} + \sum_{l=L+1}^{N} \frac{(8\times 2^{2(N-l)})}{N_p\times(l+1)^2}\right\}\boxed{\times} +$$
$$\left\{\sum_{l=L+1}^{N} \frac{(2^{2(N-l)})}{N_p\times(l+1)^2} + \sum_{l=0}^{L} \frac{(2^{2(N-l)})}{N_p\times(l+1)^2}\right\}\boxed{\div} +$$
$$\left\{\sum_{l=0}^{L} \frac{(25\times 2^{2(N-l)})}{N_p\times(l+1)^2} + \sum_{l=L+1}^{N} \frac{(25\times 2^{2(N-l)})}{N_p\times(l+1)^2}\right\}\boxed{\text{Cond}} +$$
$$(2L+1)\boxed{\text{Lock}} + (L+2)\boxed{\text{Sync}}$$

(c) Distributed memory system with $N_p$ processors:

$$
\begin{aligned}
OP_p \approx \quad & \sum_{l=0}^{L} \frac{(60\times 2^{2(N-l)})}{N_p\times(l+1)^2}\boxed{=} + \sum_{l=0}^{L} \frac{(6\times 2^{2(N-l)})}{N_p\times(l+1)^2}\boxed{+} + \\
& \sum_{l=0}^{L} \frac{(18\times 2^{2(N-l)})}{N_p\times(l+1)^2}\boxed{-} + \sum_{l=0}^{L} \frac{(8\times 2^{2(N-l)})}{N_p\times(l+1)^2}\boxed{\times} + \\
& \sum_{l=0}^{L} \frac{(2^{2(N-l)})}{N_p\times(l+1)^2}\boxed{\div} + \sum_{l=0}^{L} \frac{(25\times 2^{2(N-l)})}{N_p\times(l+1)^2}\boxed{\text{Cond}} \\
& (3L)\boxed{\text{Send}} + (3L)\boxed{\text{Receive}} \quad for \;\; p \neq 0
\end{aligned}
$$

*and*

$$
\begin{aligned}
OP_{p0} \approx \quad & \left\{\sum_{l=0}^{L} \frac{(60\times 2^{2(N-l)})}{N_p\times(l+1)^2} + \sum_{l=L+1}^{N} \frac{(60\times 2^{2(N-l)})}{N_p\times(l+1)^2}\right\}\boxed{=} + \\
& \left\{\sum_{l=0}^{L} \frac{(6\times 2^{2(N-l)})}{N_p\times(l+1)^2} + \sum_{l=L+1}^{N} \frac{(6\times 2^{2(N-l)})}{N_p\times(l+1)^2}\right\}\boxed{+} + \\
& \left\{\sum_{l=0}^{L} \frac{(18\times 2^{2(N-l)})}{N_p\times(l+1)^2} + \sum_{l=L+1}^{N} \frac{(18\times 2^{2(N-l)})}{N_p\times(l+1)^2}\right\}\boxed{-} + \\
& \left\{\sum_{l=0}^{L} \frac{(8\times 2^{2(N-l)})}{N_p\times(l+1)^2} + \sum_{l=L+1}^{N} \frac{(8\times 2^{2(N-l)})}{N_p\times(l+1)^2}\right\}\boxed{\times} + \\
& \left\{\sum_{l=0}^{L} \frac{(2^{2(N-l)})}{N_p\times(l+1)^2} + \sum_{l=L+1}^{N} \frac{(2^{2(N-l)})}{N_p\times(l+1)^2}\right\}\boxed{\div} + \\
& \left\{\sum_{l=0}^{L} \frac{(25\times 2^{2(N-l)})}{N_p\times(l+1)^2} + \sum_{l=L+1}^{N} \frac{(25\times 2^{2(N-l)})}{N_p\times(l+1)^2}\right\}\boxed{\text{Cond}} \\
& ((N_p+1)\times L)\boxed{\text{Send}} + ((N_p+1)\times L)\boxed{\text{Receive}}
\end{aligned}
$$

Pyramid reconstruction:

(a) Single processor:

$$
\begin{aligned}
OP \approx \quad & \sum_{l=0}^{N} \frac{(3\times 2^{2(N-l)})}{(l+1)^2}\boxed{=} + \sum_{l=0}^{N} \frac{(2^{2(N-l)})}{(l+1)^2}\boxed{+} + \sum_{l=0}^{N} \frac{(2^{2(N-l)})}{(l+1)^2}\boxed{\div} + \\
& \sum_{l=0}^{N} \frac{(2^{2(N-l)})}{(l+1)^2}\boxed{\text{Cond}}
\end{aligned}
$$

(b) Shared memory system with $N_P$ processors:

$$OP_p \approx \sum_{l=0}^{L} \frac{(3\times 2^{2(N-l)})}{N_p\times(l+1)^2}\boxed{=} + \sum_{l=0}^{L} \frac{(2^{2(N-l)})}{N_p\times(l+1)^2}\boxed{+} + \sum_{l=0}^{L} \frac{(2^{2(N-l)})}{N_p\times(l+1)^2}\boxed{\div} + \sum_{l=0}^{L} \frac{(2^{2(N-l)})}{N_p\times(l+1)^2}\boxed{\text{Cond}} + 1\boxed{\text{Sync}} \quad for\ p \neq 0$$

*and*

$$OP_{p0} \approx \left\{\sum_{l=0}^{L} \frac{(3\times 2^{2(N-l)})}{N_p\times(l+1)^2} + \sum_{l=L+1}^{N} \frac{(3\times 2^{2(N-l)})}{(l+1)^2}\right\}\boxed{=} + \left\{\sum_{l=0}^{L} \frac{(2^{2(N-l)})}{N_p\times(l+1)^2}\boxed{+} + \sum_{l=L+1}^{N} \frac{(2^{2(N-l)})}{(l+1)^2}\right\}\boxed{+} + \left\{\sum_{l=0}^{L} \frac{(2^{2(N-l)})}{N_p\times(l+1)^2} + \sum_{l=L+1}^{N} \frac{(2^{2(N-l)})}{(l+1)^2}\right\}\boxed{\div} + \left\{\sum_{l=0}^{L} \frac{(2^{2(N-l)})}{N_p\times(l+1)^2} + \sum_{l=L+1}^{N} \frac{(2^{2(N-l)})}{(l+1)^2}\right\}\boxed{\text{Cond}} + 1\boxed{\text{Sync}}$$

(c) Distributed memory system with $N_P$ processors:

$$OP_p \approx \sum_{l=0}^{L} \frac{(3\times 2^{2(N-l)})}{N_p\times(l+1)^2}\boxed{=} + \sum_{l=0}^{L} \frac{(2^{2(N-l)})}{N_p\times(l+1)^2}\boxed{+} + \sum_{l=0}^{L} \frac{(2^{2(N-l)})}{N_p\times(l+1)^2}\boxed{\div} + \sum_{l=0}^{L} \frac{(2^{2(N-l)})}{N_p\times(l+1)^2}\boxed{\text{Cond}} \quad for\ p \neq 0$$

*and*

$$OP_{p0} \approx \left\{\sum_{l=0}^{L} \frac{(3\times 2^{2(N-l)})}{N_p\times(l+1)^2} + \sum_{l=L+1}^{N} \frac{(3\times 2^{2(N-l)})}{(l+1)^2}\right\}\boxed{=} + \left\{\sum_{l=0}^{L} \frac{(2^{2(N-l)})}{N_p\times(l+1)^2} + \sum_{l=L+1}^{N} \frac{(2^{2(N-l)})}{(l+1)^2}\right\}\boxed{+} + \left\{\sum_{l=0}^{L} \frac{(2^{2(N-l)})}{N_p\times(l+1)^2} + \sum_{l=L+1}^{N} \frac{(2^{2(N-l)})}{(l+1)^2}\right\}\boxed{\div} + \left\{\sum_{l=0}^{L} \frac{(2^{2(N-l)})}{N_p\times(l+1)^2} + \sum_{l=L+1}^{N} \frac{(2^{2(N-l)})}{(l+1)^2}\right\}\boxed{\text{Cond}}$$

4. Perform segmentation:

(a) Single processor:

$$OP \approx \sum_{l=N}^{1} \frac{(2^{2(N-l)})}{(l+1)^2}\boxed{=} + \sum_{l=N}^{1} \frac{(2^{2(N-l)})}{(l+1)^2}\boxed{-} + \sum_{l=N}^{1} \frac{(2^{2(N-l)})}{(l+1)^2}\boxed{\text{Cond}}$$

(b) Shared memory system with $N_P$ processors:

$$OP_p \approx \sum_{l=L}^{1} \frac{(2^{2(N-l)}}{(l+1)^2}\boxed{=} + \sum_{l=L}^{1} \frac{(2^{2(N-l)}}{(l+1)^2}\boxed{-} + \sum_{l=L}^{1} \frac{(2^{2(N-l)}}{(l+1)^2}\boxed{\text{Cond}} + 2\boxed{\text{Sync}} \quad for \ p \neq 0$$

*and*

$$OP_{p0} \approx \left\{\sum_{l=N}^{L+1} \frac{(2^{2(N-l)}}{(l+1)^2} + \sum_{l=L}^{1} \frac{(2^{2(N-l)}}{(l+1)^2}\right\}\boxed{=} + \left\{\sum_{l=N}^{L+1} \frac{(2^{2(N-l)}}{(l+1)^2} + \sum_{l=L}^{1} \frac{(2^{2(N-l)}}{(l+1)^2}\right\}\boxed{-} + \left\{\sum_{l=N}^{L+1} \frac{(2^{2(N-l)}}{(l+1)^2} + \sum_{l=L}^{1} \frac{(2^{2(N-l)}}{(l+1)^2}\right\}\boxed{\text{Cond}} + 2\boxed{\text{Sync}}$$

(c) Distributed memory system with $N_P$ processors:

$$OP_p \approx \sum_{l=L}^{1} \frac{(2^{2(N-l)}}{(l+1)^2}\boxed{=} + \sum_{l=L}^{1} \frac{(2^{2(N-l)}}{(l+1)^2}\boxed{-} + \sum_{l=L}^{1} \frac{(2^{2(N-l)}}{(l+1)^2}\boxed{\text{Cond}} + 1\boxed{\text{Receive}} \quad for \ p \neq 0$$

*and*

$$OP_{p0} \approx \left\{\sum_{l=N}^{L+1} \frac{(2^{2(N-l)}}{(l+1)^2} + \sum_{l=L}^{1} \frac{(2^{2(N-l)}}{(l+1)^2}\right\}\boxed{=} + \left\{\sum_{l=N}^{L+1} \frac{(2^{2(N-l)}}{(l+1)^2} + \sum_{l=L}^{1} \frac{(2^{2(N-l)}}{(l+1)^2}\right\}\boxed{-} + \left\{\sum_{l=N}^{L+1} \frac{(2^{2(N-l)}}{(l+1)^2} + \sum_{l=L}^{1} \frac{(2^{2(N-l)}}{(l+1)^2}\right\}\boxed{\text{Cond}} + (N_p - 1)\boxed{\text{Sends}}$$

5. Scale the resulting image:

(a) Single processor:

$$OP \approx 3 \times 2^{2N}\boxed{=} + 2^{2N}\boxed{\times} + 2 \times 2^{2N}\boxed{-} + 2^{2N}\boxed{\div} + 2 \times 2^{2N}\boxed{\text{Cond}}$$

(b) Shared memory system with $N_p$ processors:

$$OP \approx \frac{3\times 2^{2N}}{N_p}\boxed{=} + \frac{2^{2N}}{N_p}\boxed{\times} + \frac{2\times 2^{2N}}{N_p}\boxed{-} + \frac{2^{2N}}{N_p}\boxed{\div} + \frac{2\times 2^{2N}}{N_p}\boxed{\text{Cond}} + 3\boxed{\text{Sync}}$$

(c) Distributed memory system with $N_p$ processors:

$$OP_p \approx \frac{3\times 2^{2N}}{N_p}\boxed{=} + \frac{2^{2N}}{N_p}\boxed{\times} + \frac{2\times 2^{2N}}{N_p}\boxed{-} + \frac{2^{2N}}{N_p}\boxed{\div} + \frac{2\times 2^{2N}}{N_p}\boxed{\text{Cond}} + 2\boxed{\text{Send}} + 2\boxed{\text{Receive}} \quad for \ p \neq 0$$

$$OP_{p_0} \approx \frac{3\times 2^{2N}}{N_p}\boxed{=} + \frac{2^{2N}}{N_p}\boxed{\times} + \frac{2\times 2^{2N}}{N_p}\boxed{-} + \frac{2^{2N}}{N_p}\boxed{\div} +$$
$$\frac{2\times 2^{2N}}{N_p}\boxed{\text{Cond}} + (2N_p - 2)\boxed{\text{Send}} + (2N_p - 2)\boxed{\text{Receive}}$$

It can be inferred from the above calculations that the most important factor in determining process time reduction is the ratio $\frac{2^N}{NP}$. The higher the ratio, the higher the utilization for both shared and distributed memory systems for the 5 steps. The obvious reason is that for a larger ratio there are more rows for each processor to work on; therefore, there is an increase in utilization. There is a decrease in utilization when the number of rows becomes smaller, and there is more overhead involved in communication for distributed memory systems, and the main overhead for shared memory systems are memory contention and synchronization. The processors in both architectures become idle after performing on the last row. This effect causes lower utilization as $\frac{2^N}{NP}$ gets smaller.

The performance and process utilization of both algorithms were measured by processing a 128 by 128 pixel image. Figure 4.1 shows processor utilization for the shared memory algorithm considering 2, 4, 8, 16, 32 processors and compares utilization. It can be seen that for increasing number of processors greater than four, utilization degrades because of memory contention overhead. Figure 4.2 shows processor utilization for the distributed memory algorithm. Utilization decreases as the number of processors increases. It can be seen that for 32 processors the utilization approaches zero. This indicates that the communication overhead for the distributed memory algorithm has more effect on processor utilization than memory contention for shared memory systems as shown in Figure 4.3. Figure 4.4 shows the speed up of both algorithms for pyramid generation (step 1). Both algorithms showed improved performance as the number of processors is increased. For 4, 8, 16 processors, both algorithms showed improved per-

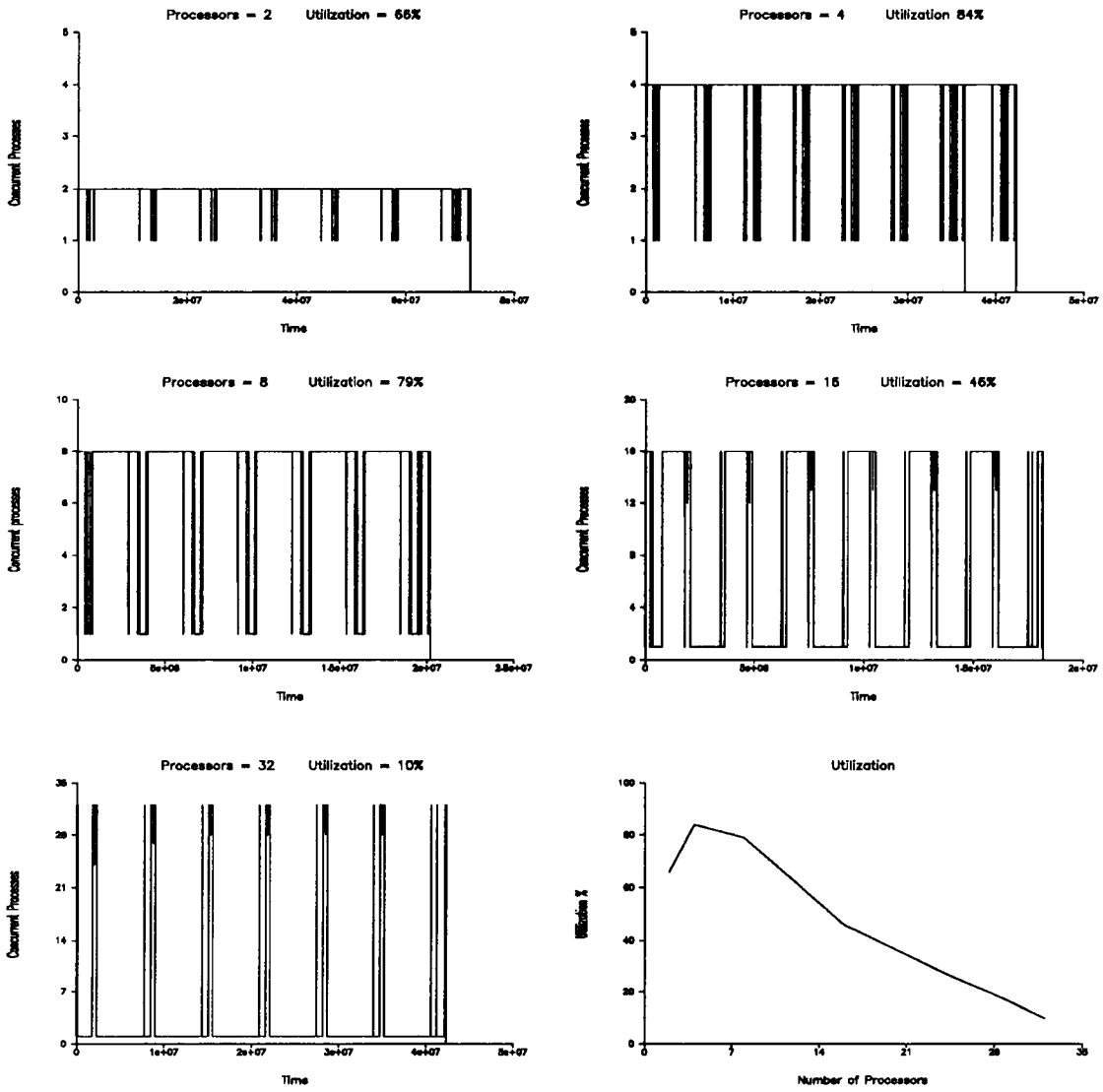



Figure 4.1: Processor utilization for shared memory system for $N = 7$.

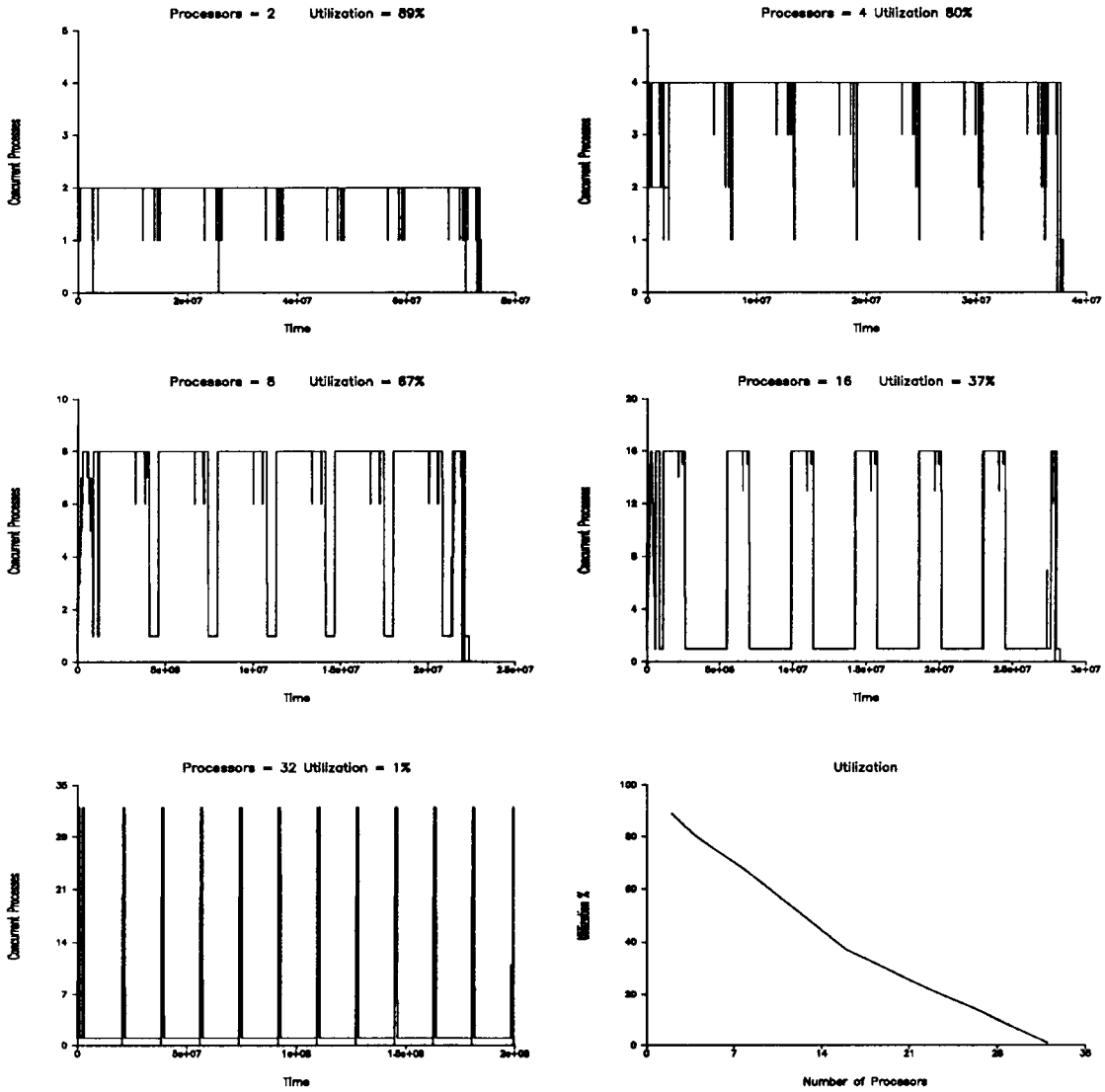



Figure 4.2: Processor utilization for distributed memory system.

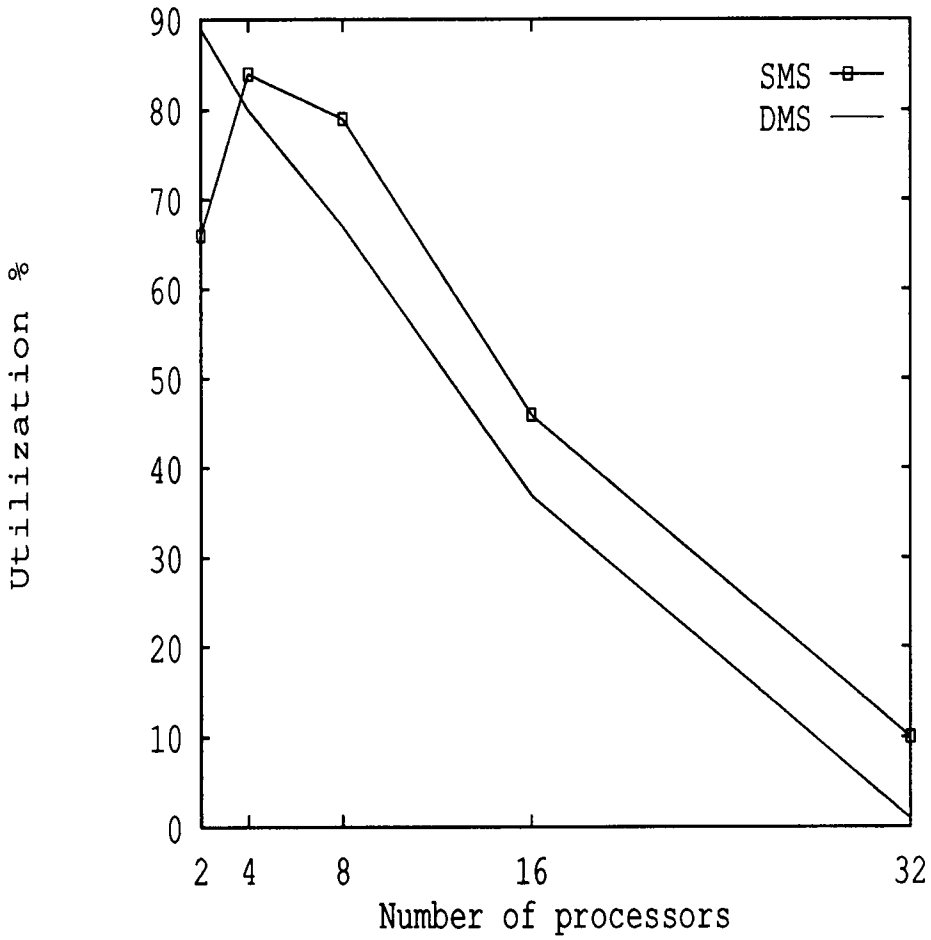



Figure 4.3: Utilization graphs for SMS and DMS algorithms.

formance as the number of processors increased; however the shared memory algorithm showed better speed up. For 32 processors, both algorithms showed a decrease in speed up; however, the speed up for the shared memory algorithm decreased at a lower rate than for the distributed memory algorithm. Figure 4.5 shows the speed up of both algorithms for computing the mean of the base image (step 2). The graphs show a linear speed up for step 2. Figure 4.6 shows the speed up for 6 iterationS of fuzzy link process. For the shared memory algorithm for 2, 4 , 8, 16 processors, speed up increases, and it decreases for 32 processors. For the distributed memory algorithm there is a speed up for number of processors 2, 4, 8 and a decrease for speed up for processors 16, 32. The shared memory algorithm performs better for fuzzy link iteration because it has less memory contention overhead than communication overhead for the distributed memory algorithm. Figures 4.7 and 4.8 show the same results for speed up as shown in Figure 4.4, Figure 4.5 and Figure 4.6. Finally 4.9 shows the speed up for both algorithms. For 2, 4 processors both algorithms perform equally. For 8 processors the shared memory algorithm shows a better speed up. For 16 processors the shared memory algorithm showS a greater decrease in speed up. For 32 processors, the distributed memory algorithm shows almost no speed up. This indicates that as the number of processors increases the communication overhead takes more effect on the performance of the algorithm for the distributed memory algorithm, and for the shared memory algorithm memory contention has less effect on the fuzzy pyramid image segmentation algorithm.

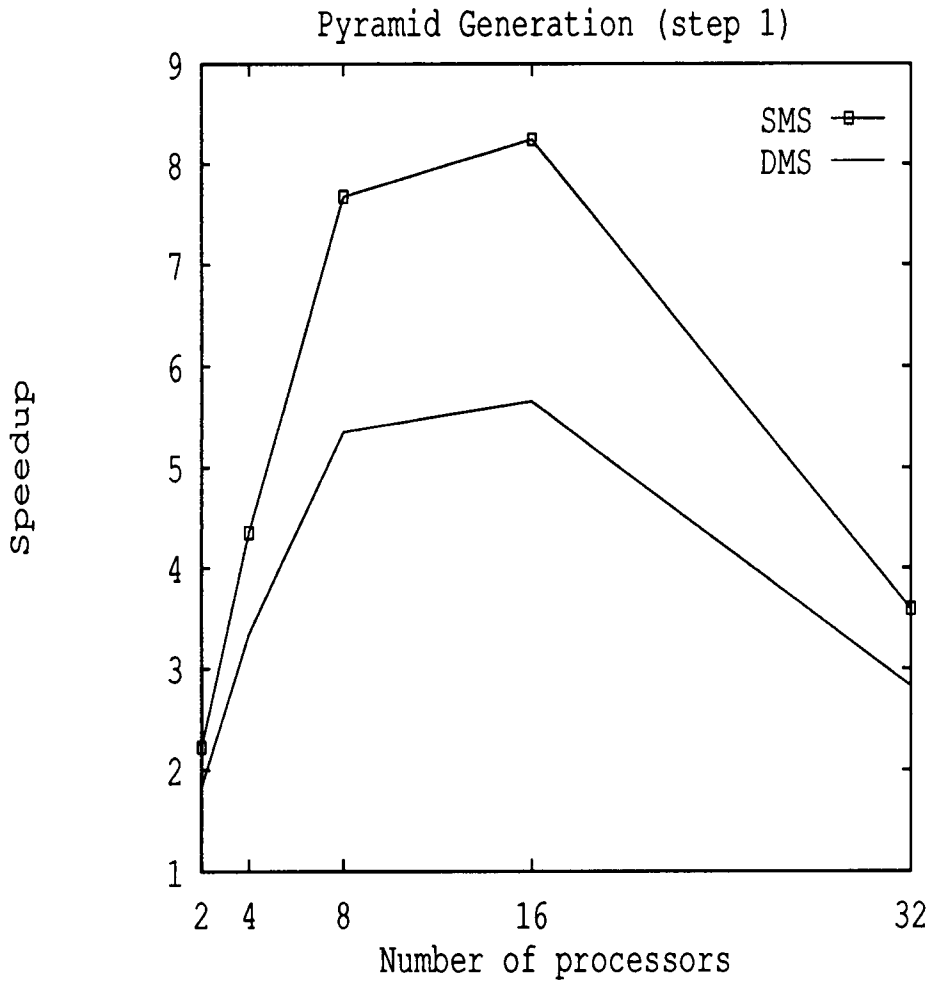



Figure 4.4: Speed up graphs for step 1 of SMS and DMS algorithms.

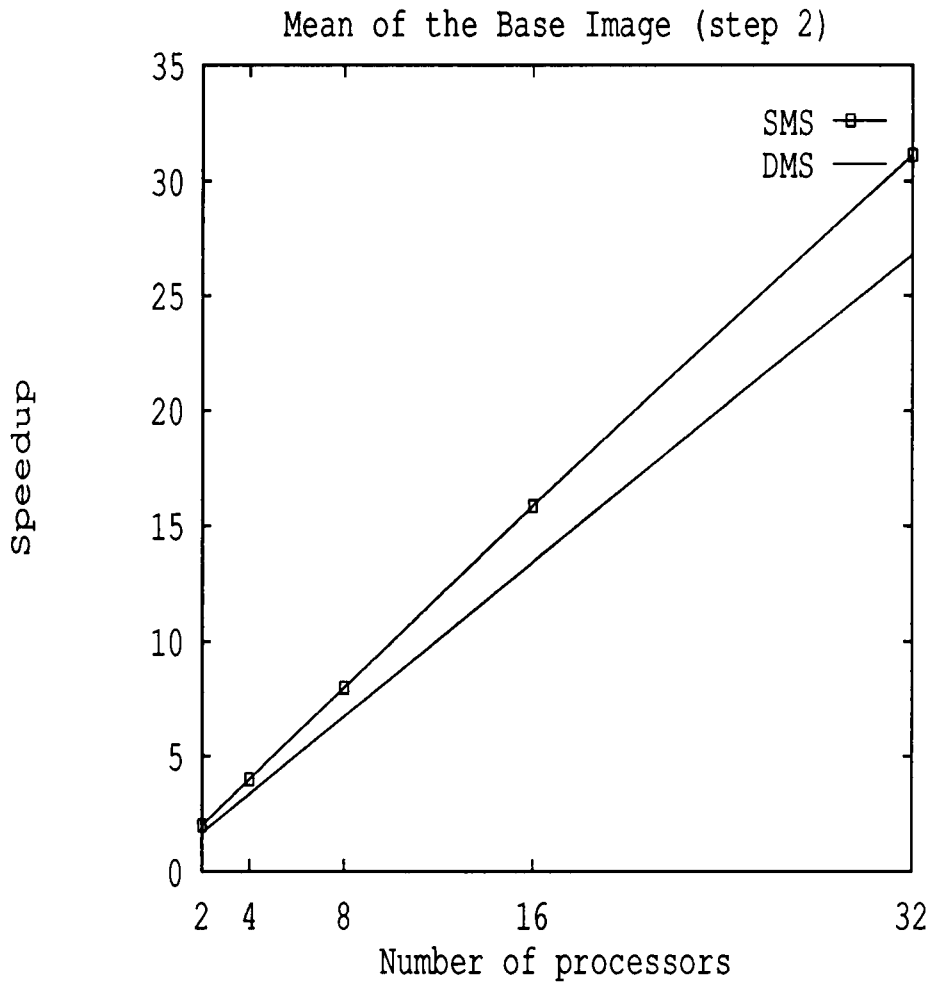



Figure 4.5: Speed up graphs for step 2 of SMS and DMS algorithms.

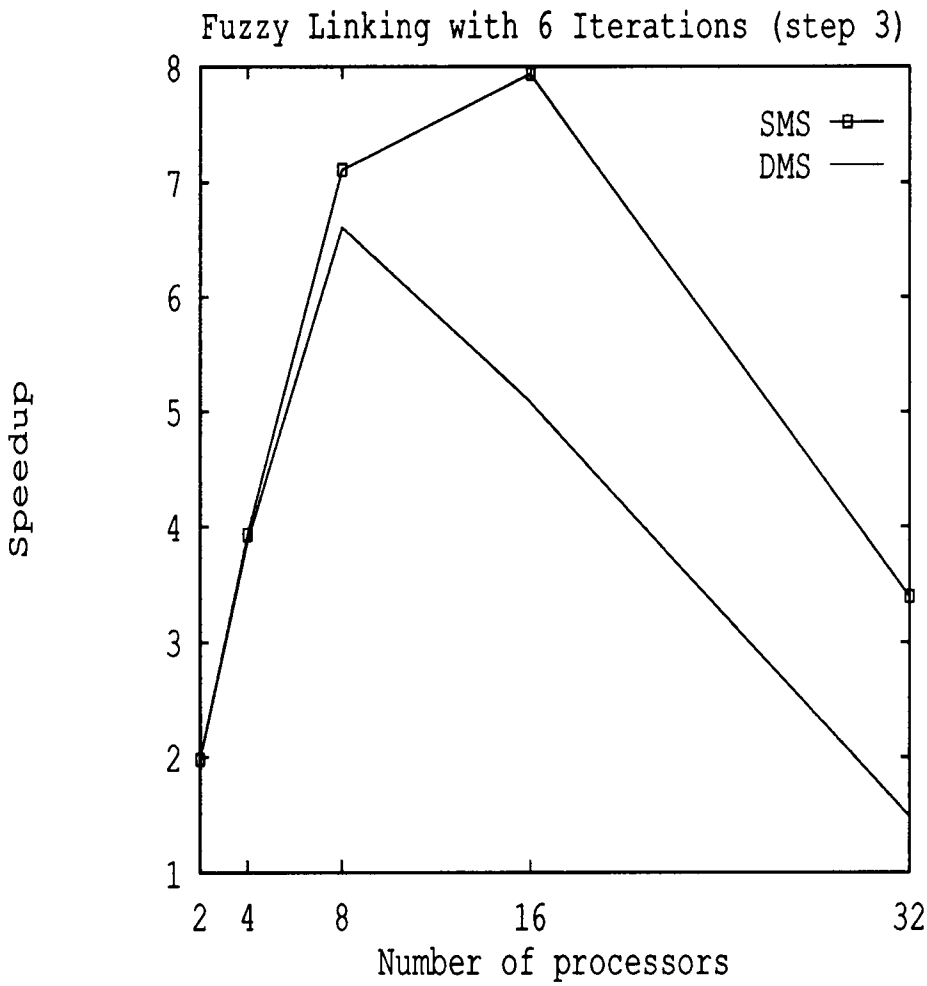



Figure 4.6: Speed up graphs for step 3 of SMS and DMS algorithms.

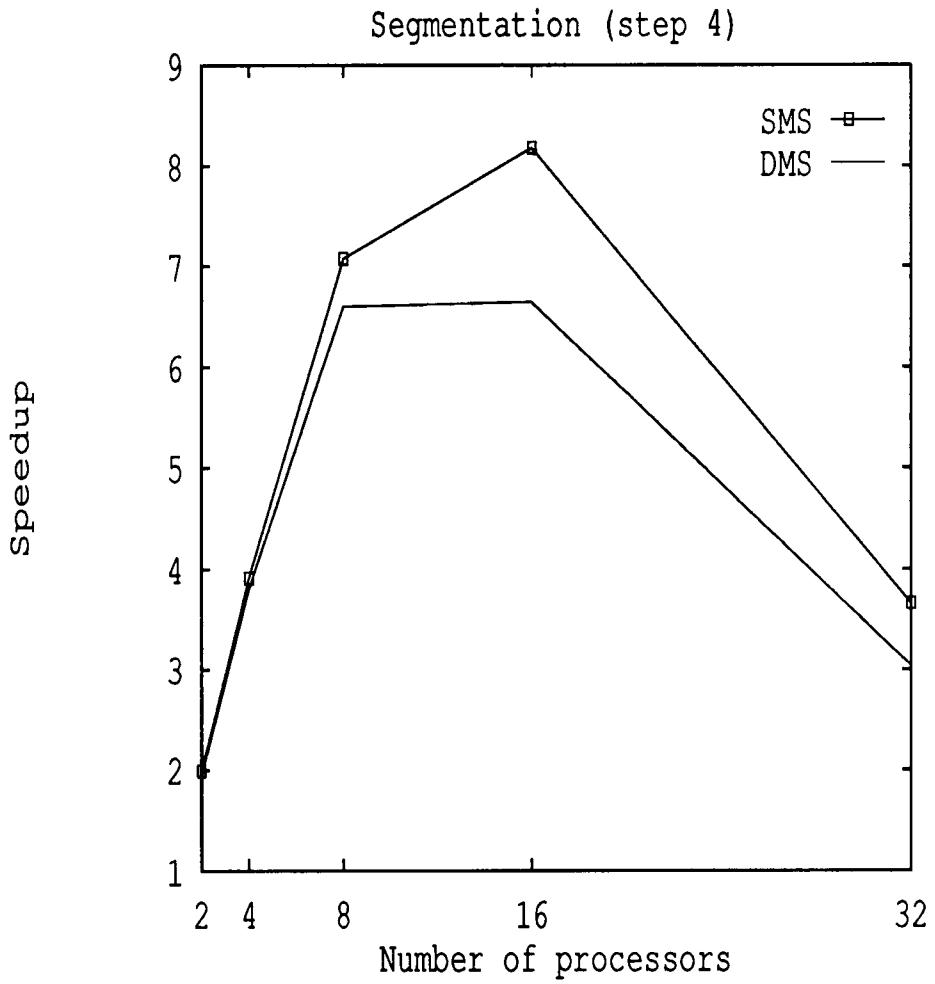



Figure 4.7: Speed up graphs for step 4 of SMS and DMS algorithms.

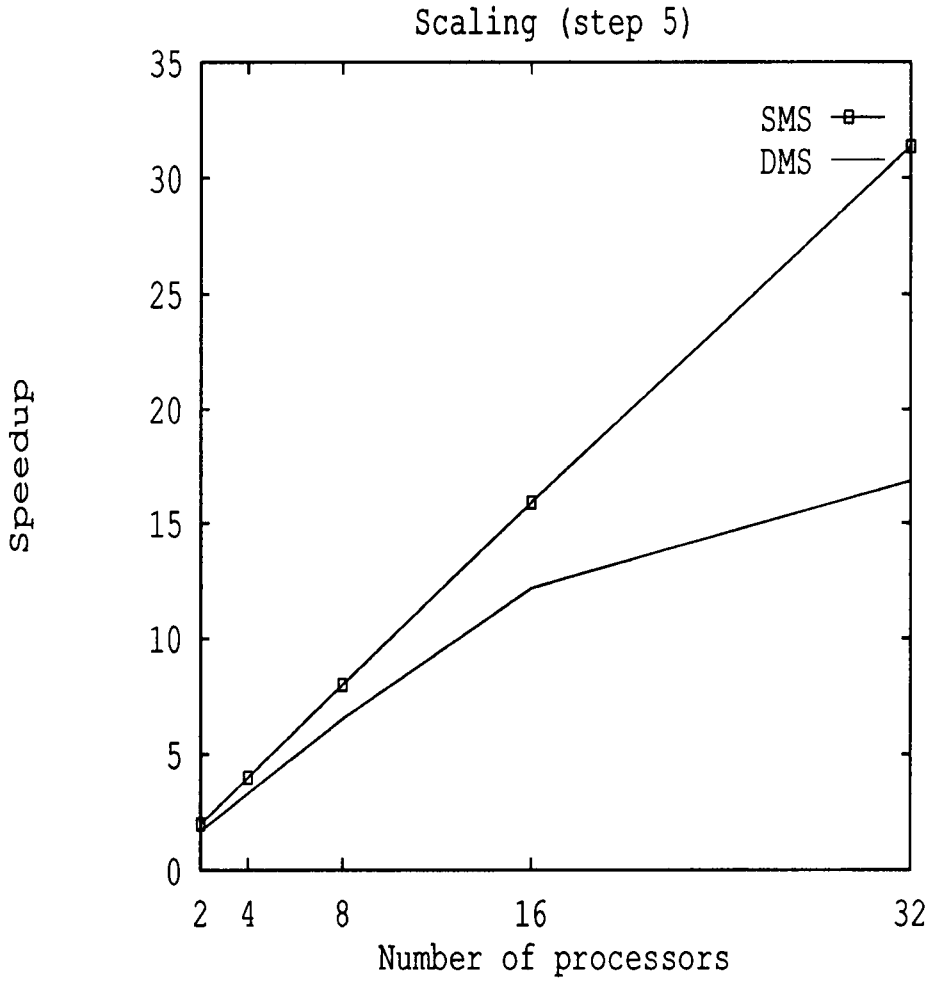



Figure 4.8: Speed up graphs for step 5 of SMS and DMS algorithms.

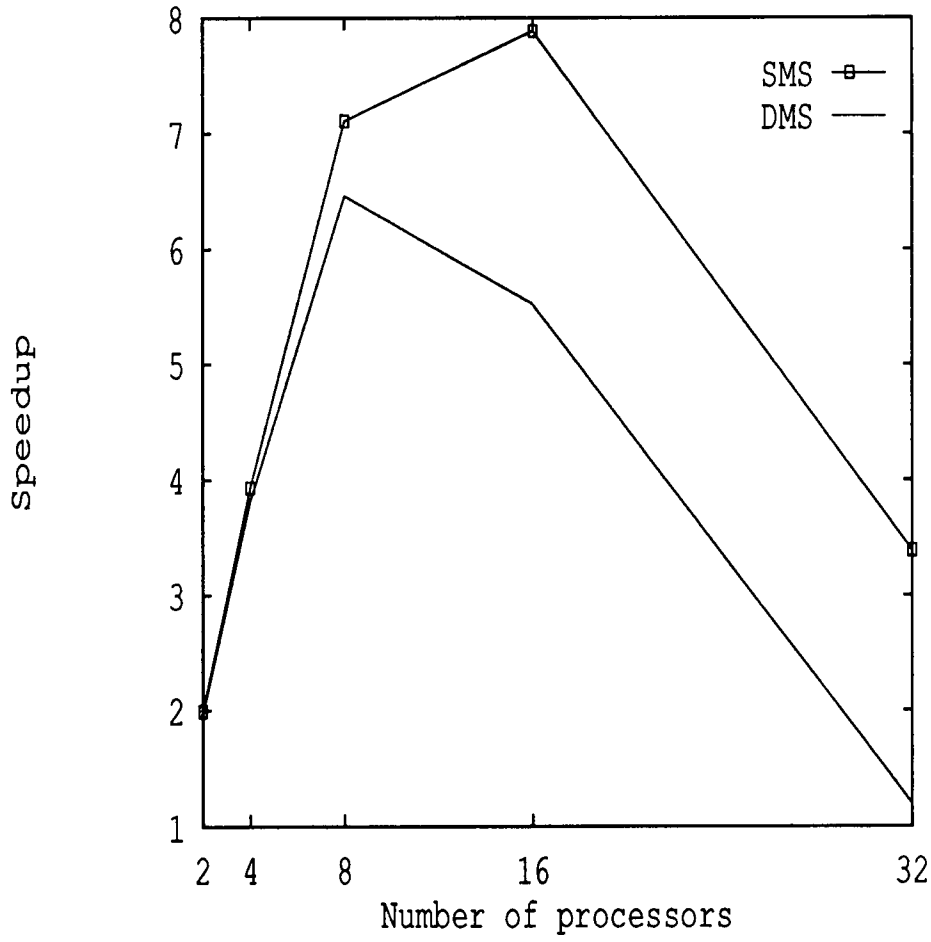



Figure 4.9: Speed up graphs for all steps of SMS and DMS algorithms.

# CHAPTER 5

# CONCLUSIONS

The parallel image segmentation algorithm using the shared memory systems speeds up the process of image segmentation by dividing the base image among the available processors. However, this can only happen if there is synchronization and effective memory management among the processors. This algorithm is designed to perform image segmentation in parallel by using synchronization and memory management.

The parallel image segmentation algorithm using the distributed memory systems also speeds up the process of image segmentation by dividing the base image among the available processors, but it differs in overheads and techniques used. Its disadvantage lies in communication overhead. The processors need to have access to the memories of other processors in their system. This algorithm facilitates and speeds communication between processors.

Both algorithms showed speed up for the process of image segmentation. However, the shared memory algorithm is more efficient because memory management overhead has less effect on the performance than communication overhead for the distributed memory algorithm.

# BIBLIOGRAPHY

# BIBLIOGRAPHY


[1] Almasi S. G., and Gottlieb A., "Highly Parallel Computing," BenjaminCummings, Menlo Park, Calif., pp. 12-29, 1989.

[2] Antonisse H. J., "Image Segmentation in Pyramids," *Computer Graphics and Image Processing*, vol. 19, pp. 367-383, 1982.

[3] Atallah M., and Goodrich M. T., "Efficient Parallel Solutions to Some Geometric Problems," *Journal of Parallel and Distributed Computing*, vol. 3, pp. 492-507, 1986.

[4] Brzakovic D., Beck H., and Sufi N., "An Approach to Defect Detection in Materials Characterized by Complex Textures," *Pattern Recognition*, pp. 99-147, 1989.

[5] Burt P. J., "The Pyramid as a Structure for Efficient Computation," *Multiresolution Image Processing and Analysis*, (Ed. A. Rosenfield) pp. 6-35, Springer-Verlag, 1984.

[6] Burt P. J., Hong T., and Rosenfield A., "Segmentation and Estimation of Image Region Properties Through Cooperative Hierarchical Computation," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-11, pp. 802-809, Dec. 1981.

[7] Chen P. C., and Pavlidis T., "Image Segmentation as an Estimation Problem," *Computer Graphics and Image Processing*, vol. 12, pp. 153-172, 1980.

[8] Cibulskis J., and Dyer C. R., "Node Linking Strategies in Pyramids for Image Segmentation," in *Multiresolution Image Processing and Analysis*, (Ed. A. Rosenfield), pp. 109-120, Springer-Verlag, 1984.

[9] Cuerra C., and Hambrusch S., "Parallel Algorithms for Line Detection on a Mesh," *Journal of Parallel and Distributed Computing*, vol. 6, pp. 1-19, 1989.

[10] Cvetanovic Z., "The Effects of Problem Partitioning, Allocation, and Granularity on the Performance of Multiple-Processor Systems," *IEEE Transactions on Computers*, vol. C-36, no. 4, pp. 421-432, 1987.

[11] Cybenko G., "Dynamic Load Balancing for Distributed Memory Multiprocessors," *Journal of Parallel and Distributed Computing*, vol. 7, pp. 279-301, 1989.

[12] Denning P. J., "Parallel Computing and its Evolution," *Communications of the ACM*, vol. 29, no. 12, pp. 1163-1167, Dec. 1986.

[13] Dongara J. J., and Duff I., "Advanced Architecture Computers," Technical Memorandum no. 57, Argonne National Laboratory, Mathematics and Computer Science Division, Argonne, Illinois, Jan. 1988.

[14] Dunigan T. H., " Performance of Three Hypercubes," Technical Report ORNL/TM-10400, Oak Ridge National Laboratory, Oak Ridge, TN, April 1987.

[15] Foldes S., "A Characterization of Hypercubes," *Discrete Mathematics*, vol. 17, pp. 155-159, 1977.

[16] Frenkel K. A., "Evaluating Two Massively Parallel Machines," *Communications of the ACM*, vol. 29, no. 8, pp. 753-758, 1986.

[17] Guerra C., and Hambrush S., "Parallel Algorithms for Line Detectio on a Mesh," *Journal of Parallel and Distributed Computing*, vol. 6, pp. 1-19, 1989.

[18] Hoare C. A. R., "Communicating Sequential Processes," *Communications of the ACM*, vol. 21, no. 8, pp. 666-677, 1978.

[19] Hwang K., Tseng P., and Kim D., "An Orthogonal Multiprocessor for Parallel Scientific Computations," *IEEE Transactions on Computers*, vol. 38, no. 1, pp. 47-61, 1989.

[20] Kim K., and Kumar V. K. P., "Parallel Memory Systems for Image Processing," *IEEE Proceeding on Computer Vision and Pattern Recognition*, San Diego, CA, pp. 654-659, 1989.

[21] Liu W., Yeh T., Batchelor W. E., and Cavin R., "Bit Level Concurrency in Real-Time Geometric Feature Extraction," *IEEE Proceeding on Computer Vision and Pattern Recognition*, Ann Arbor, MI, pp. 957-962, 1988.

[22] Liu Y., and Wang C., "Analysis of Prioritized Multiprocessor Systems," *Journal of Parallel and Distributed Computing*, vol. 7, pp. 321-334, 1989.

[23] Lu M., and Varman P., "Optimal Algorithms for Rectangle Problems on a Mesh-Connected Computer," *Journal of Parallel and Distributed Computing*, vol. 5, pp. 131-153, 1988.

[24] McBryan O. A. , "Parallel Computing: Architectures and Software," invited address, Comp-euro 87 Conference, Hamburg, May 11, 1987.

[25] Miller R., and Stout Q. F., "Simulating Essential Pyramids," *IEEE Proceeding on Computer Vision and Pattern Recognition*, Ann Arbor, MI, pp. 912-917, 1988.

[26] Mudge T.N., and Abdel-Rahman T.S., "Vision Algorithms for Hypercube Machines," *Journal of Parallel and Distributed Computing*, vol. 4, pp. 79-94, 1987.

[27] Perez A. "Parallel Segmentation of Range Images on a Hypercube-Connected Distributed Computer," PhD Dissertation, University of Tennessee, Knoxville, Dec. 1989.

[28] Pfister G. F., and Norton V. A., "Hot Spot Contention and Combining in Multistage Interconnection Networks," *IEEE Transaction on Computers*, vol. C-34, no. 10, pp. 943-948, 1985.

[29] Quin J. M., "Designing Efficient Algorithms for Parallel Computers," New York, pp. 20-80, 1987.

[30] Rettberg R., and Thomas R., "Contention is no Obstacle to Shared-Memory Multiprocessing," *Communications of the ACM*, vol. 29, no. 12, pp. 1202-1212, Dec. 1986.

[31] Rosenfield A., "Quadtrees and Pyramids for Pattern Recognition and Image Processing," *IEEE Proc. 5th Intl. Conf. on Pattern Recognition*, Miami Beach, FL, pp. 802-811, 1980.

[32] Saad Y., and Schultz M. H., "Topological Properties of Hypercube," Technical Report YALEU/DCS/RR-389, Yale University, Department of Computer Science, New Haven, CT, June 1985.

[33] Salim A. G., *The Design and Analysis of Parallel Algorithms*, Englewood Cliffs, N.J., pp. 4-33, 1989.

[34] Seitz C. L., "The Cosmic Cube," *Communications of the ACM*, vol. 28 pp. 22-33, 1985.

[35] Satori-Angus A., "Fuzzy Segmentation," Tech. Rep. TR-EE-86-16, Electrical Engineering Department, University of Tennessee, Knoxville, TN, Jan. 1986.

[36] Staut Q. F., "Supporting Divide-and-Conquer Alorithms for Image Processing," *Journal of Parallel and Distributed Computing*, vol. 4, pp. 95-115, 1987.

[37] Sufi N. "Fuzzy Pyramid Linking," M.S. Thesis, University of Tennessee, Knoxville, Dec. 1989.

[38] Tanimoto S. L., and Pavlidis T., "A Hierarchial Data Structure for Picture Processing," *Computer Graphics and Image Processing*, vol. 4, pp. 104-119, 1975.

[39] Tanimoto S. L., "Pictorial Feature Distortion in a Pyramid," *Computer Graphics and Image Processing*, vol. 5, pp. 333-352, 1978.

[40] Wittie L. D., "Communication Structures for Large Networks of Microcomputers," *IEEE Transactions on Computers*, vol. C-30, no. 4, pp. 264-273, April 1981.

[41] Wiejak J. S., Buxton H., and Buxton B. F., "Convolution with Separable Masks for Early Image Processing," *Computer Vision, Graphics, and Image Processing*, vol. 32, pp. 279-290, 1985.

[42] Willebeek-LeMair M., and Reeves A. P., "Region Growing On a Hypercube Multiprocessor," *Communications of the ACM*, vol. 33, no. 11, pp. 1033-1042, 1988.

[43] Yonezawa A., *ABCL an Object-Oriented Concurrent System*, MIT Press, Cambridge, Mass., pp. 15-40, 1990.

[44] Zadeh L. A., "Fuzzy Sets," *Information and Control*, vol. 8, pp. 338-353, 1965.

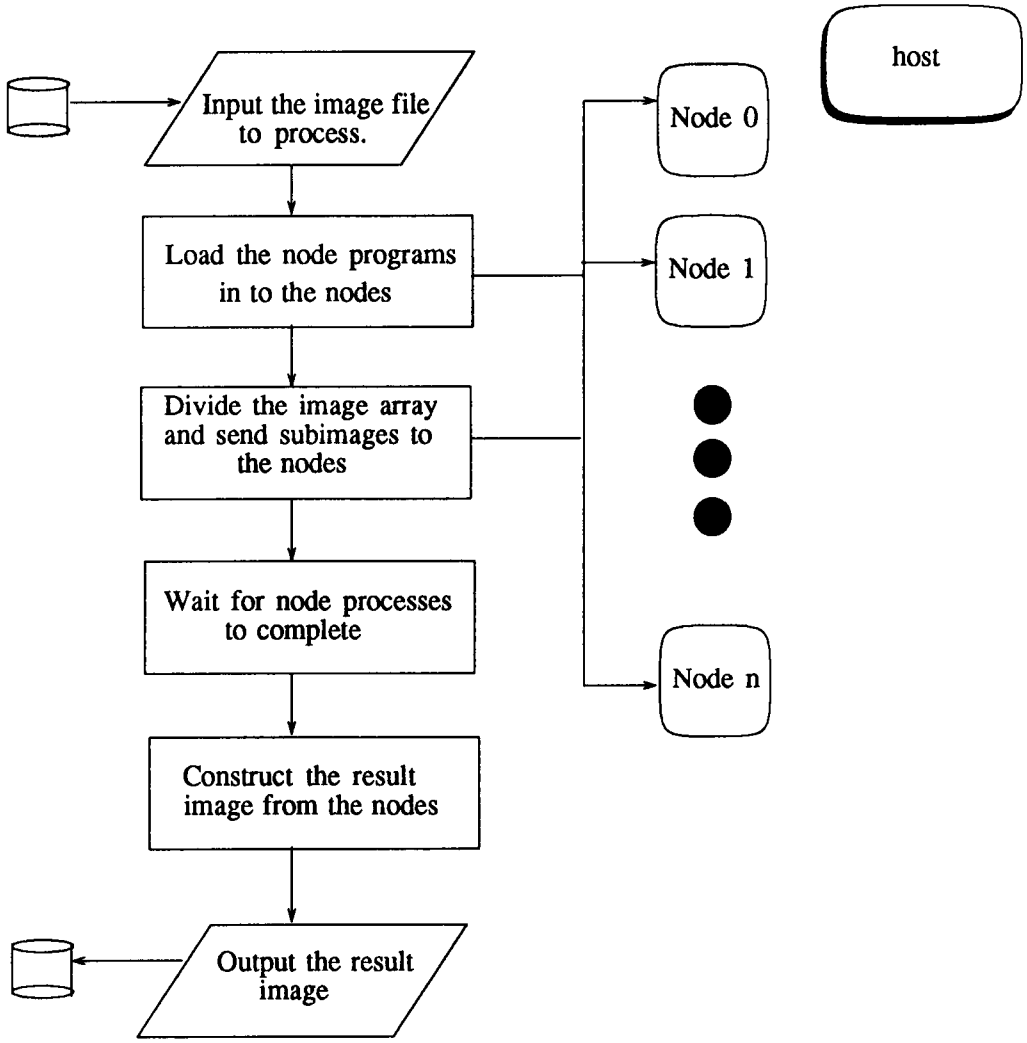
[45] Zadeh L. A., "Quantative Fuzzy Semantics," *Information Sciences*, vol. 3, pp. 159-176, 1971.

# APPENDICES

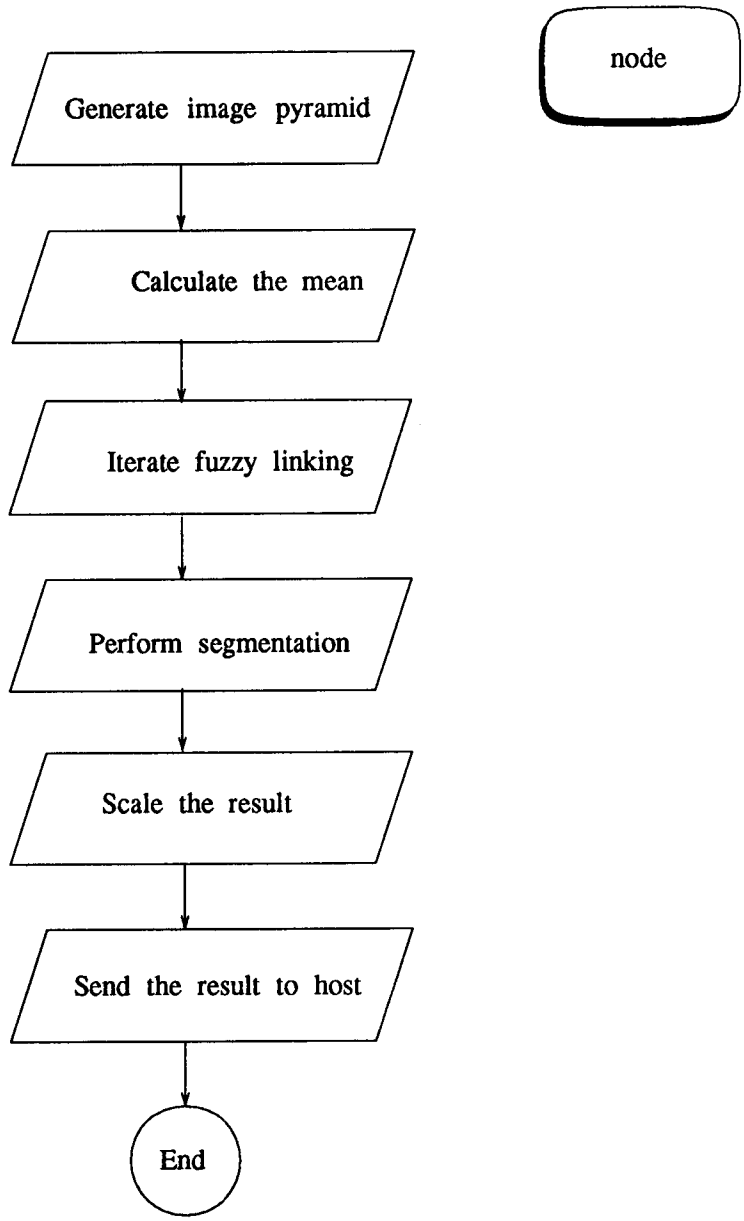
# APPENDIX A

# PYRAMID SEGMENTATION FOR DISTRIBUTED MEMORY SYSTEMS

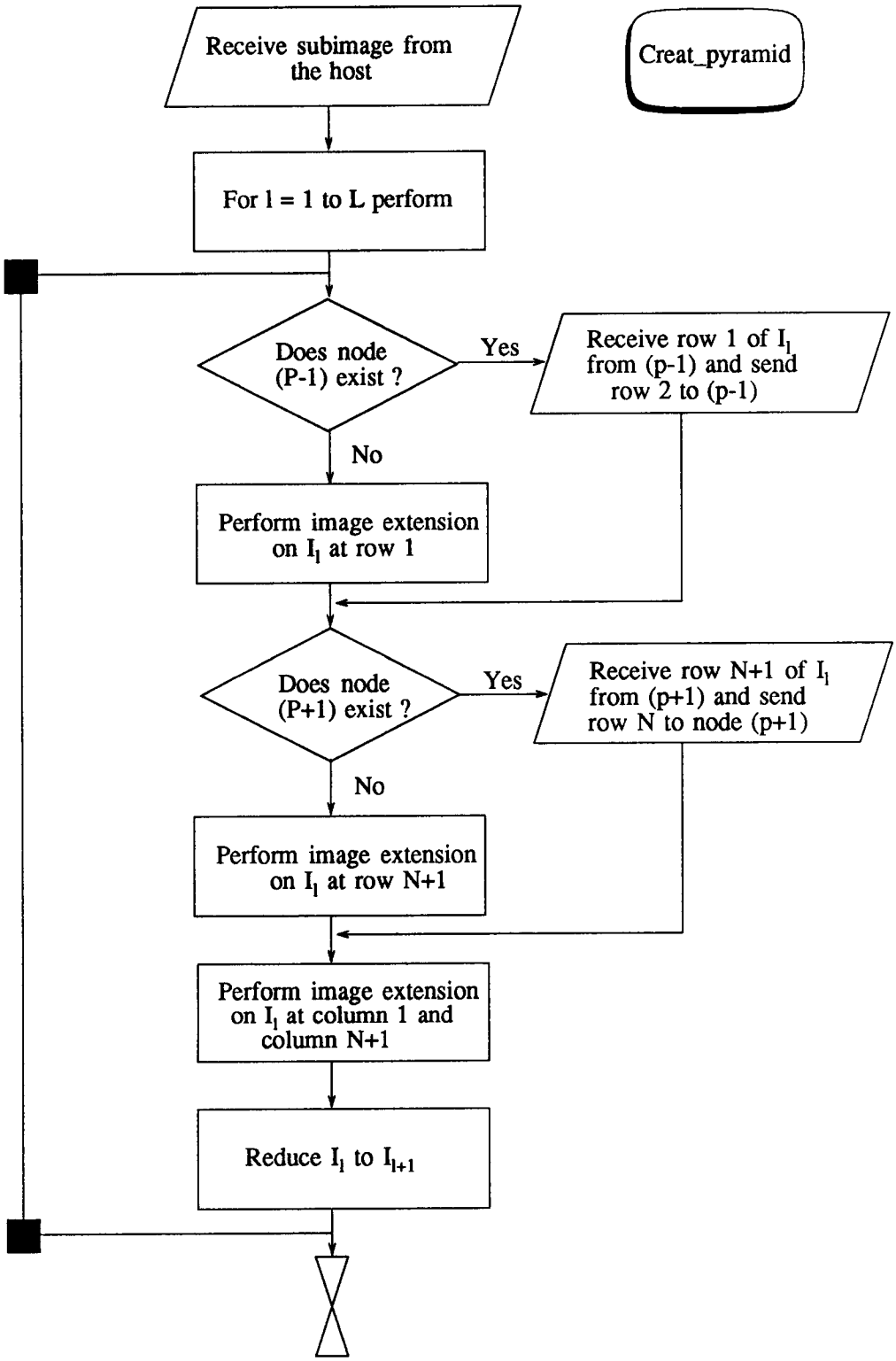
## A.1 Host Program

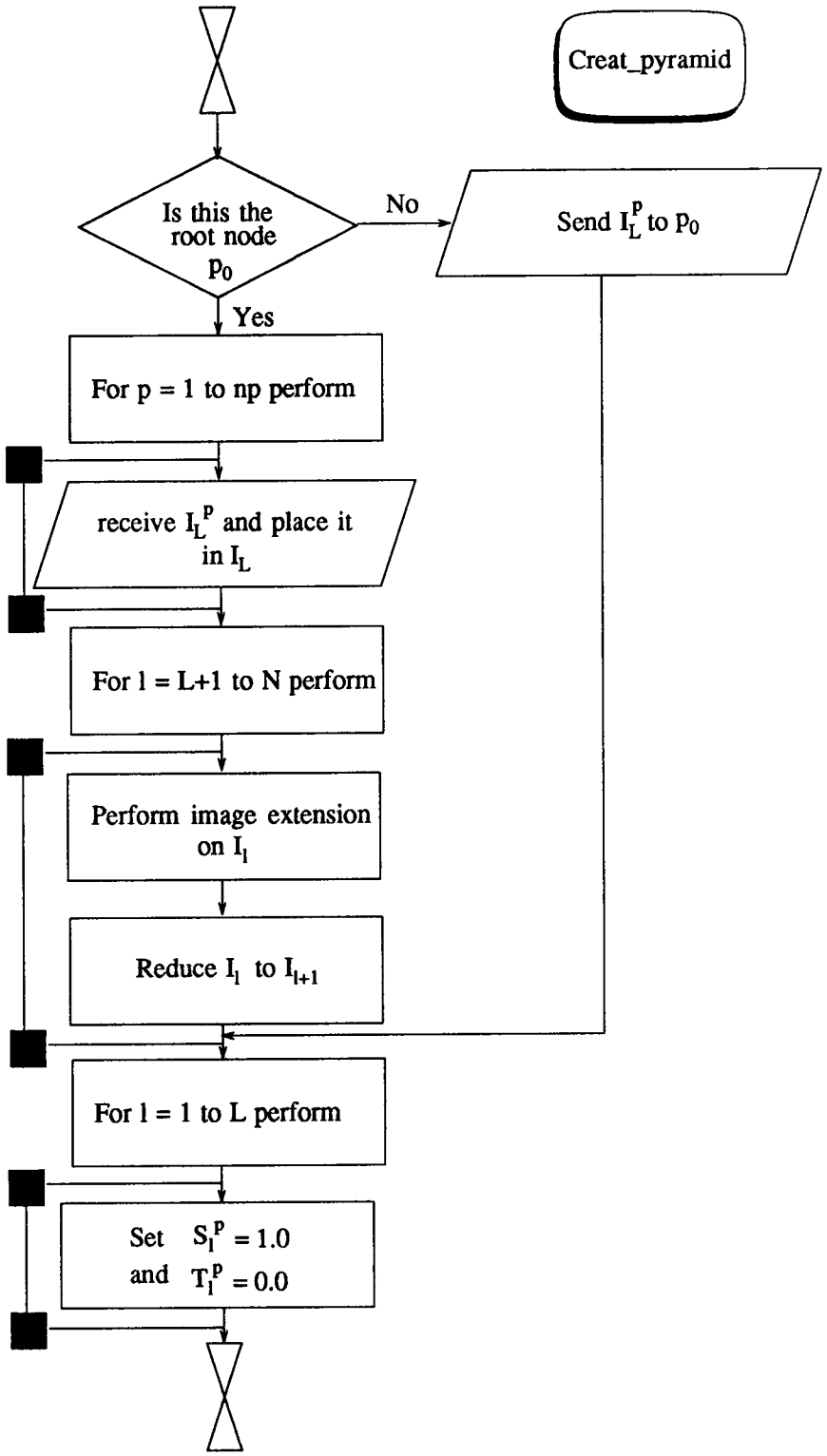


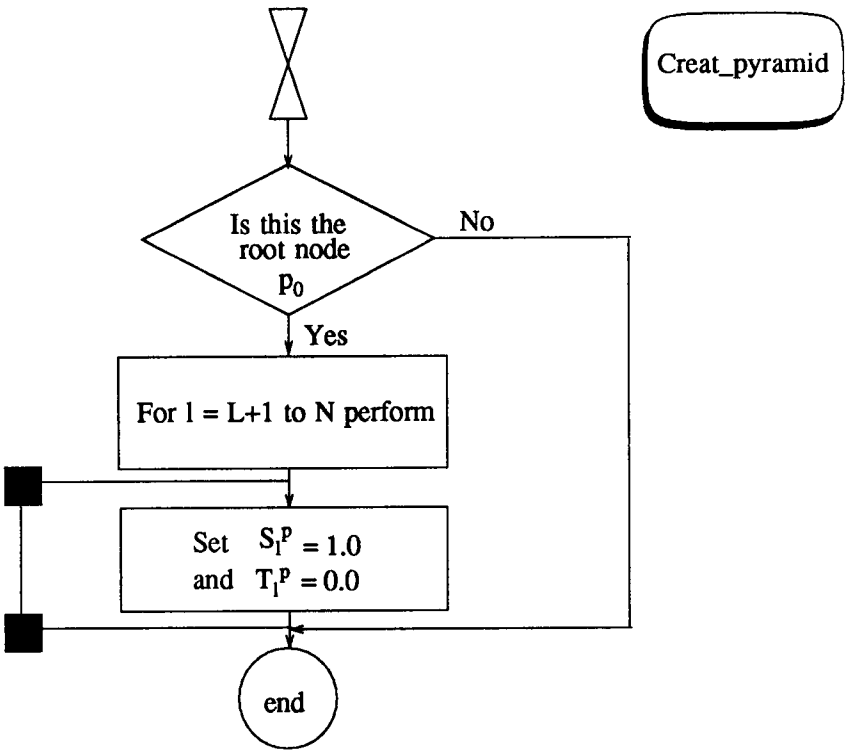
## A.2 Node Program



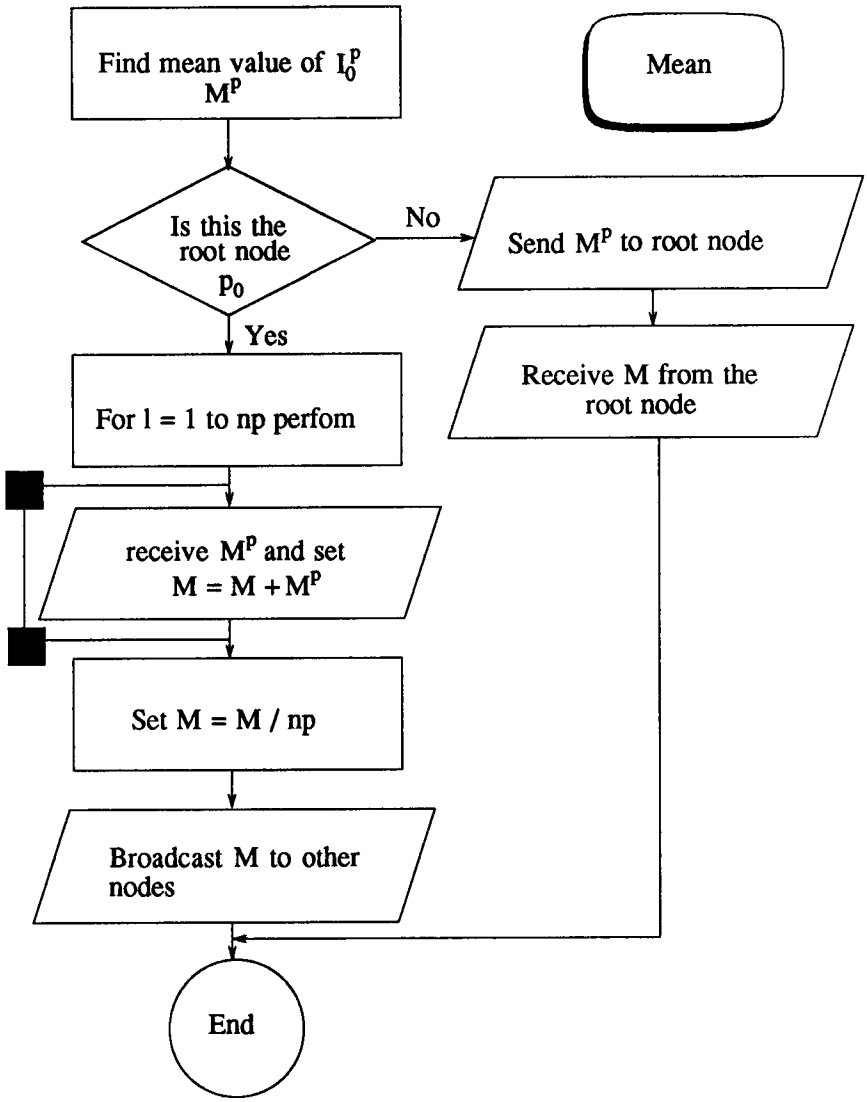
### A.2.1 Pyramid Creation



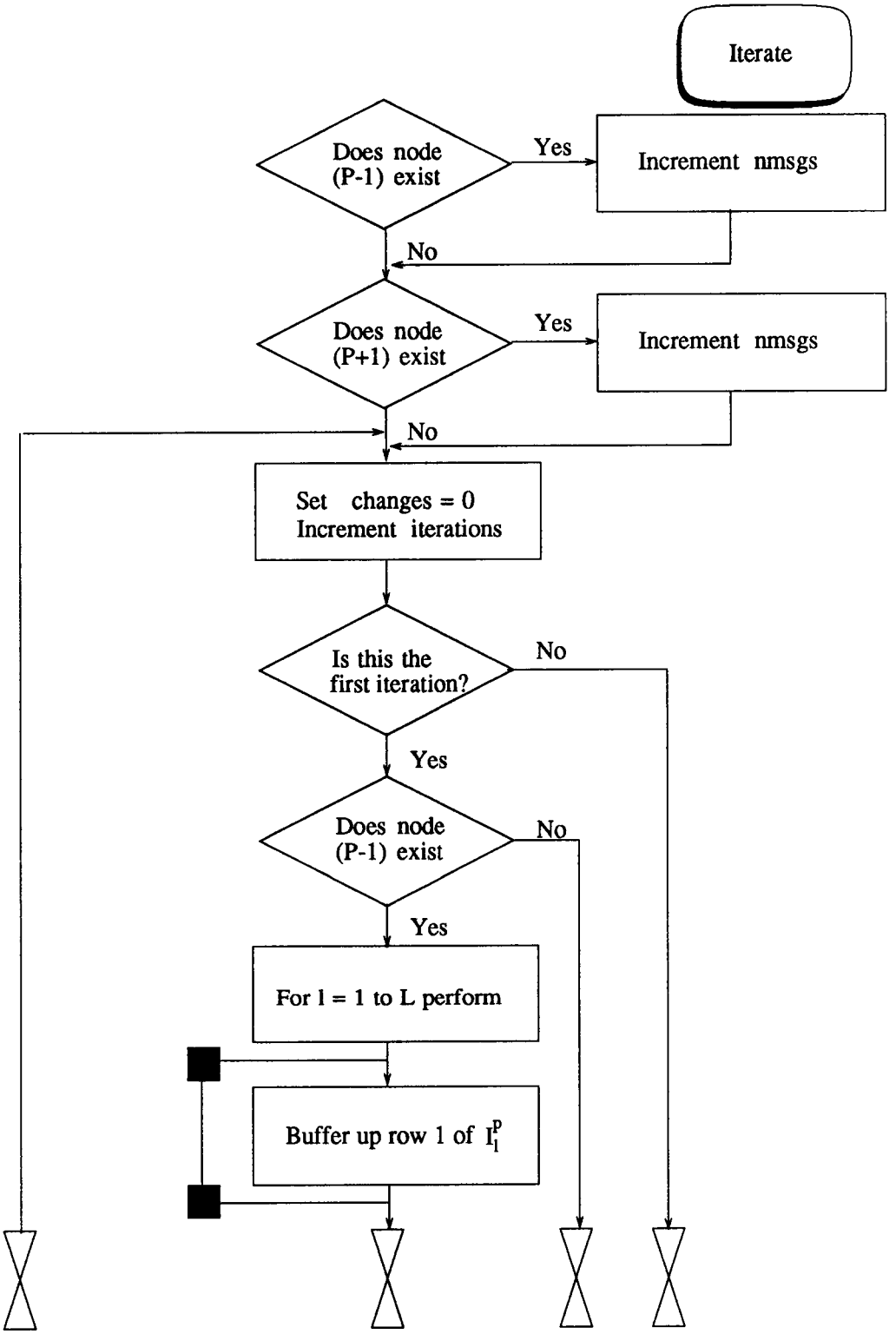

Creat_pyramid
Is this the root node $p_0$
No
Send $I_L^p$ to $p_0$
Yes
For p = 1 to np perform
receive $I_L^p$ and place it in $I_L$
For l = L+1 to N perform
Perform image extension on $I_l$
Reduce $I_l$ to $I_{l+1}$
For l = 1 to L perform
Set $S_l^p = 1.0$ and $T_l^p = 0.0$

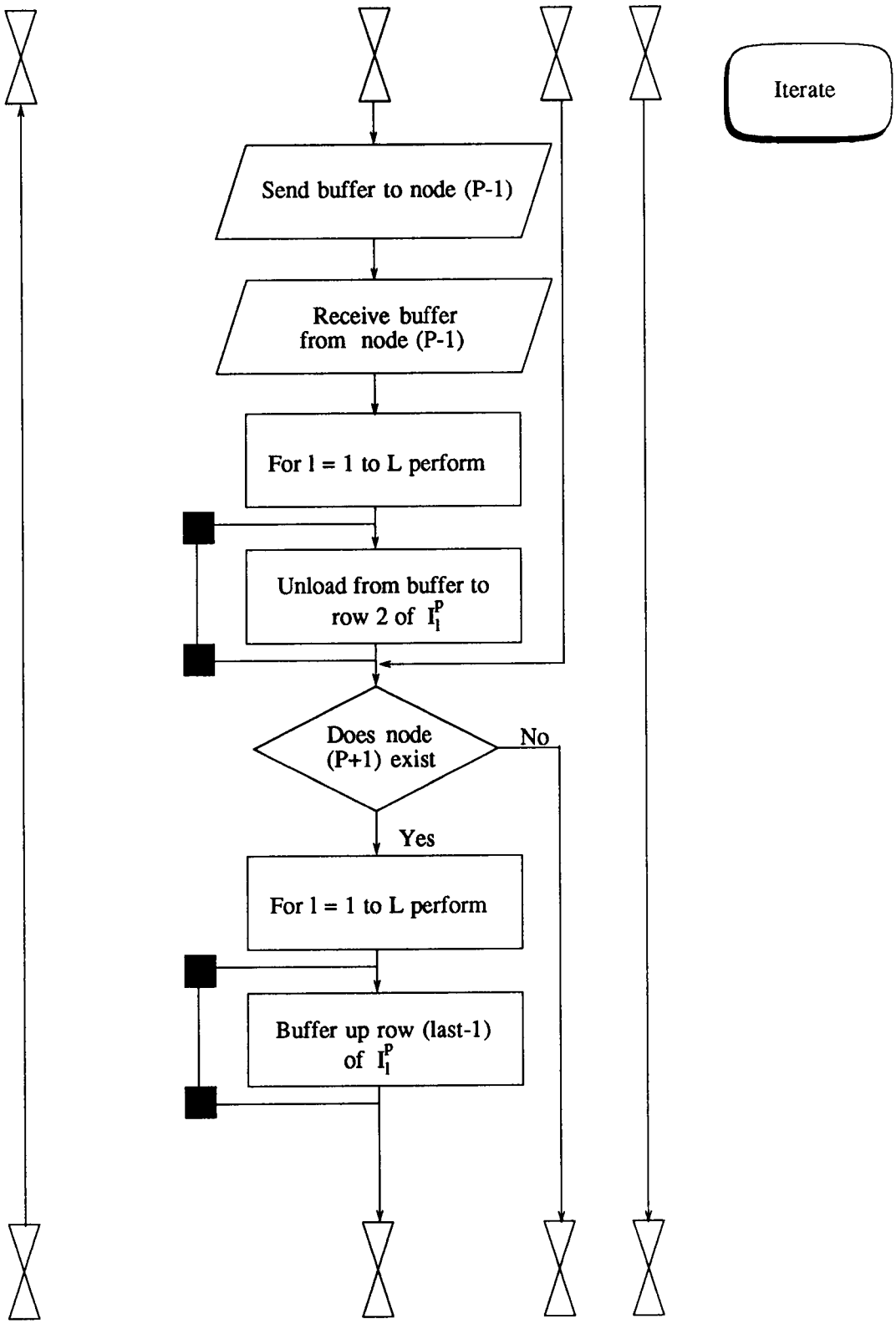

Creat_pyramid
Is this the root node $p_0$
No
Yes
For l = L+1 to N perform
Set $S_l^p = 1.0$ and $T_l^p = 0.0$
end

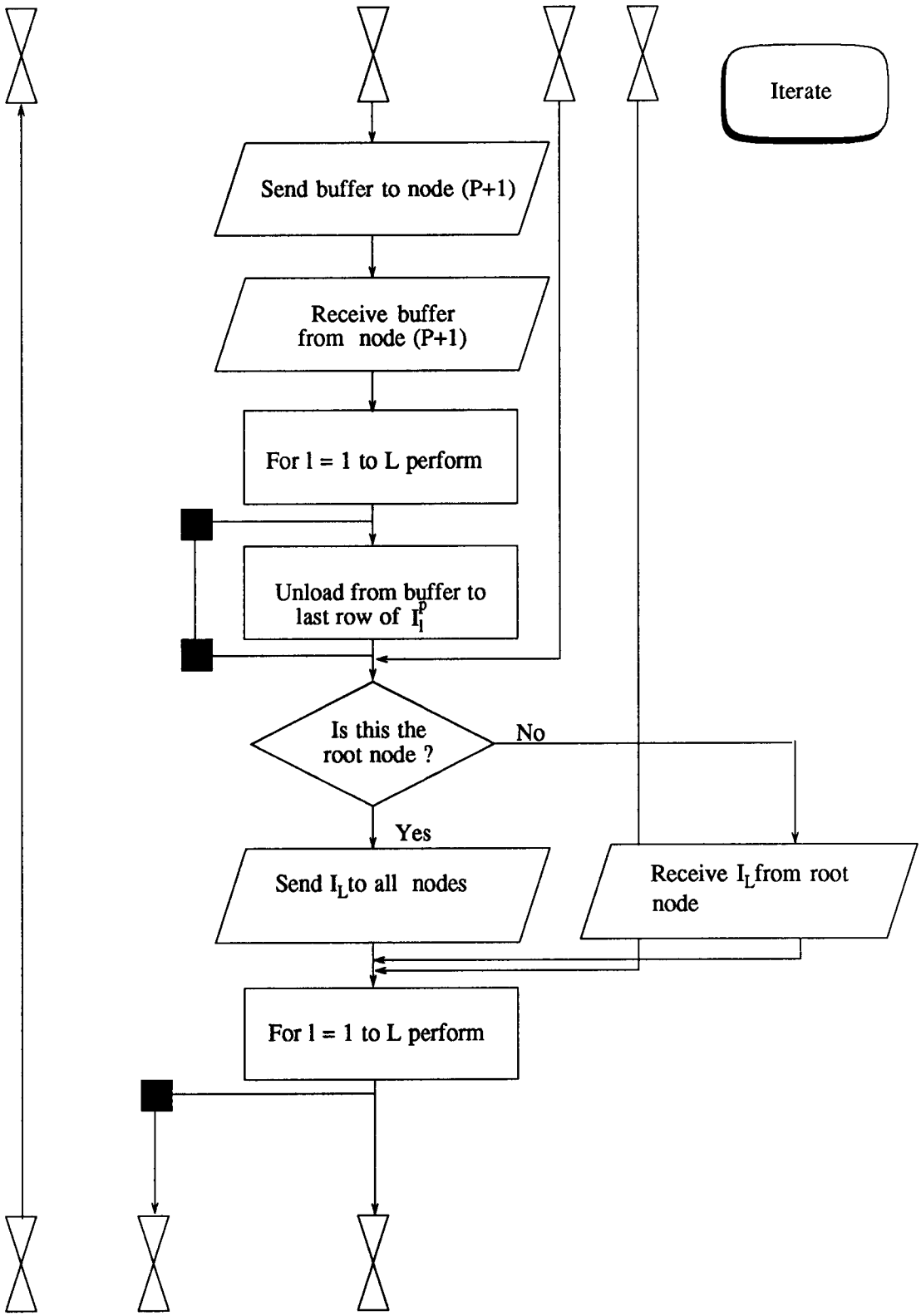
### A.2.2 Finding the Mean Value of Base of Pyramid

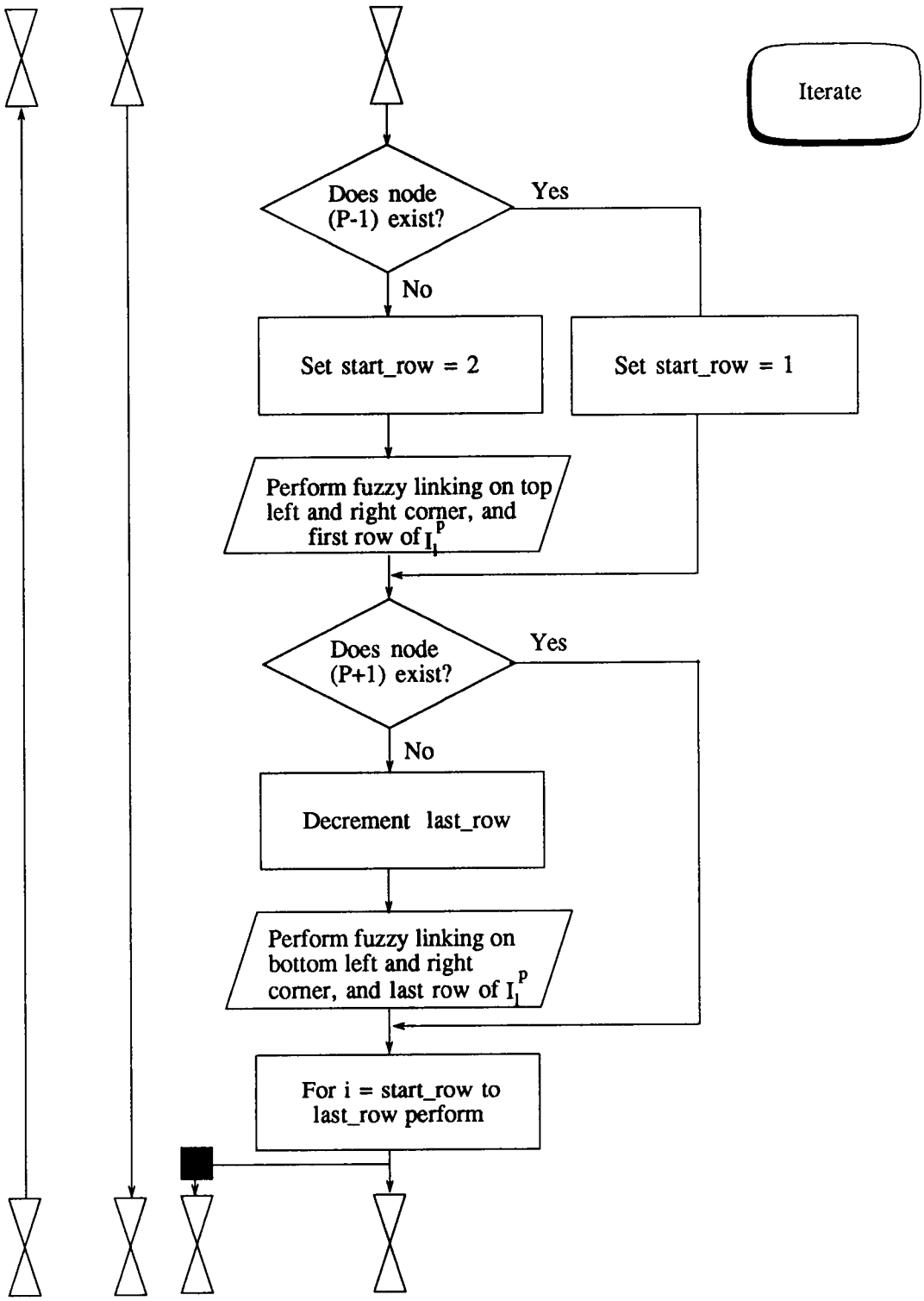


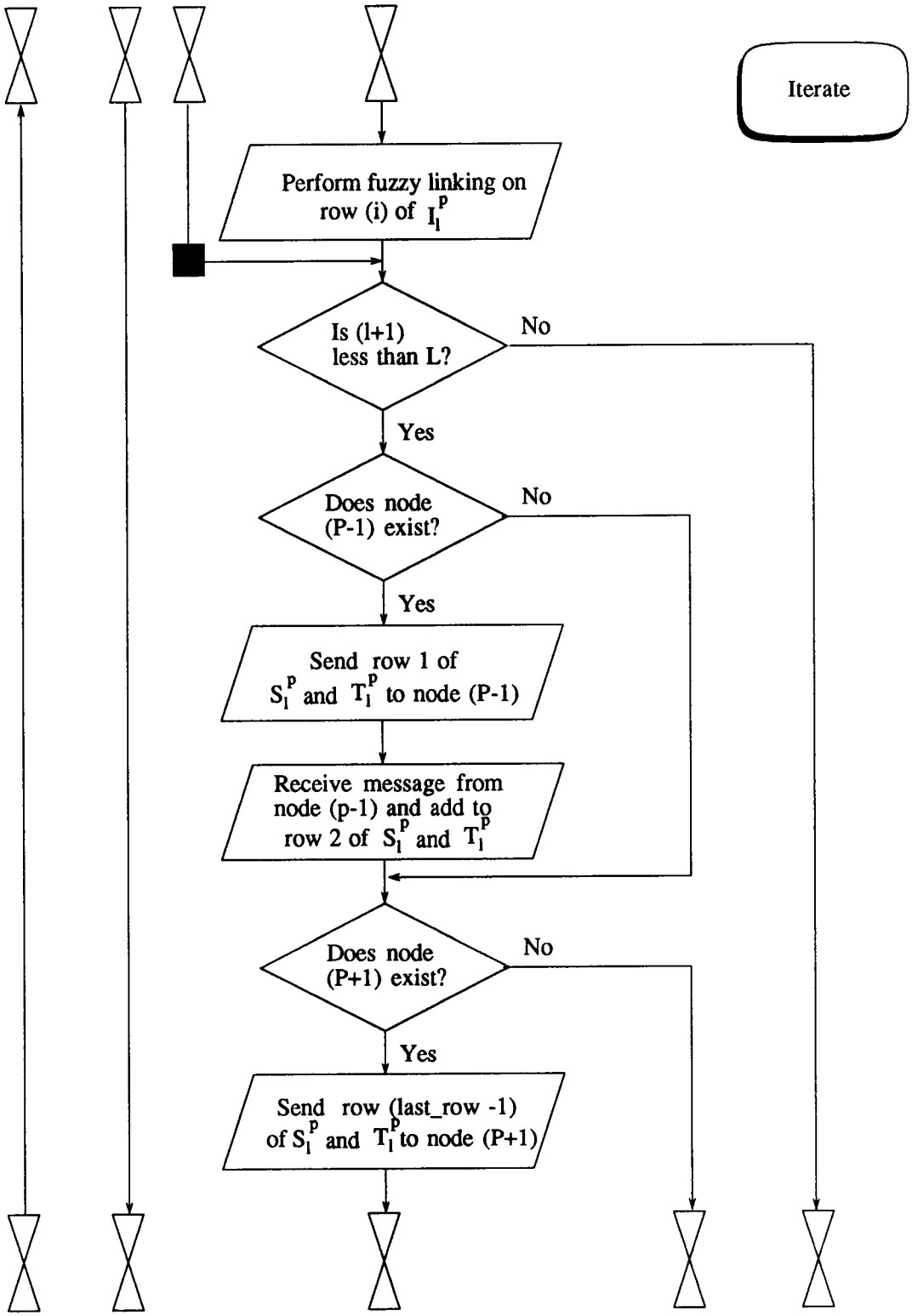
### A.2.3 Linking Process

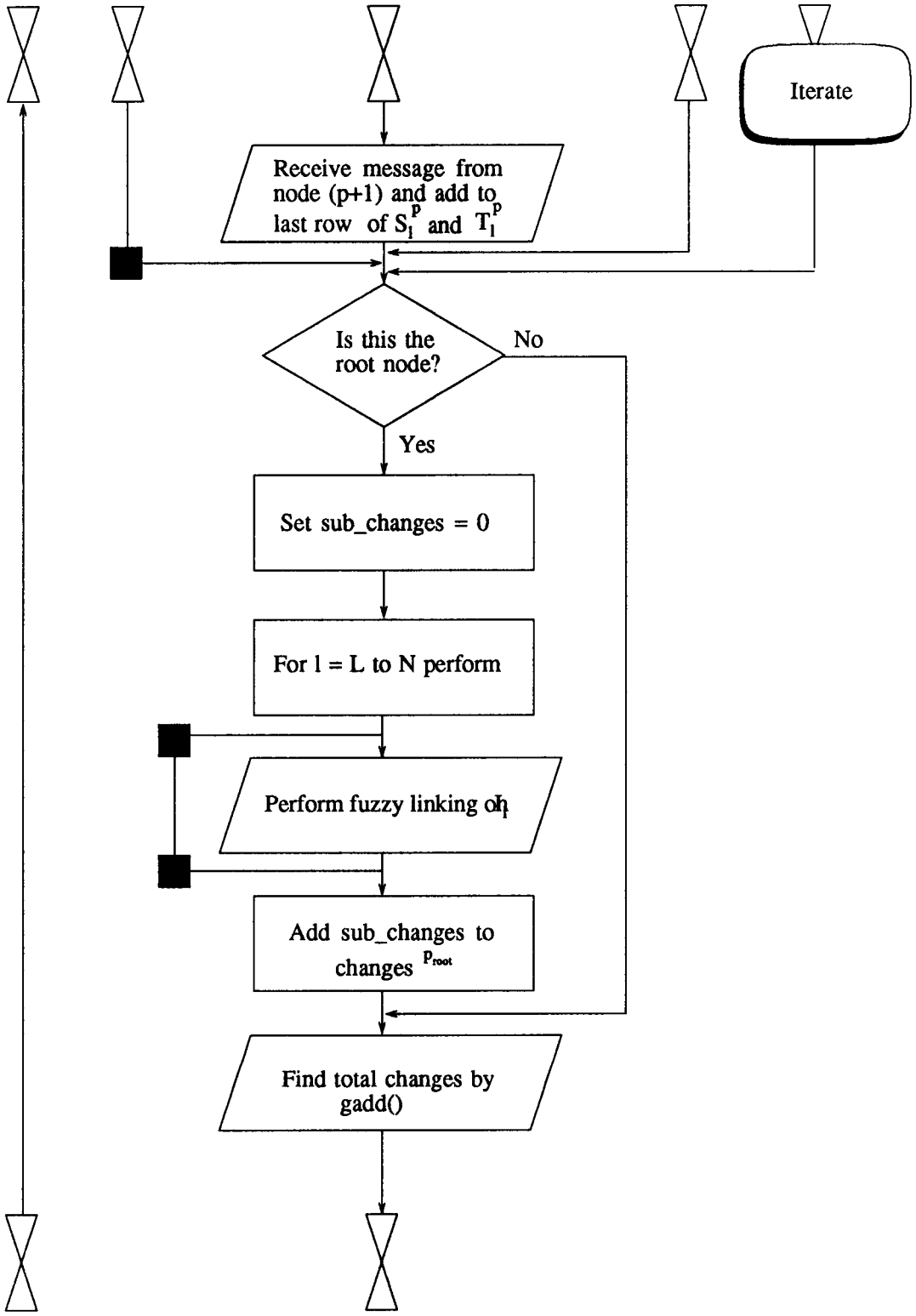


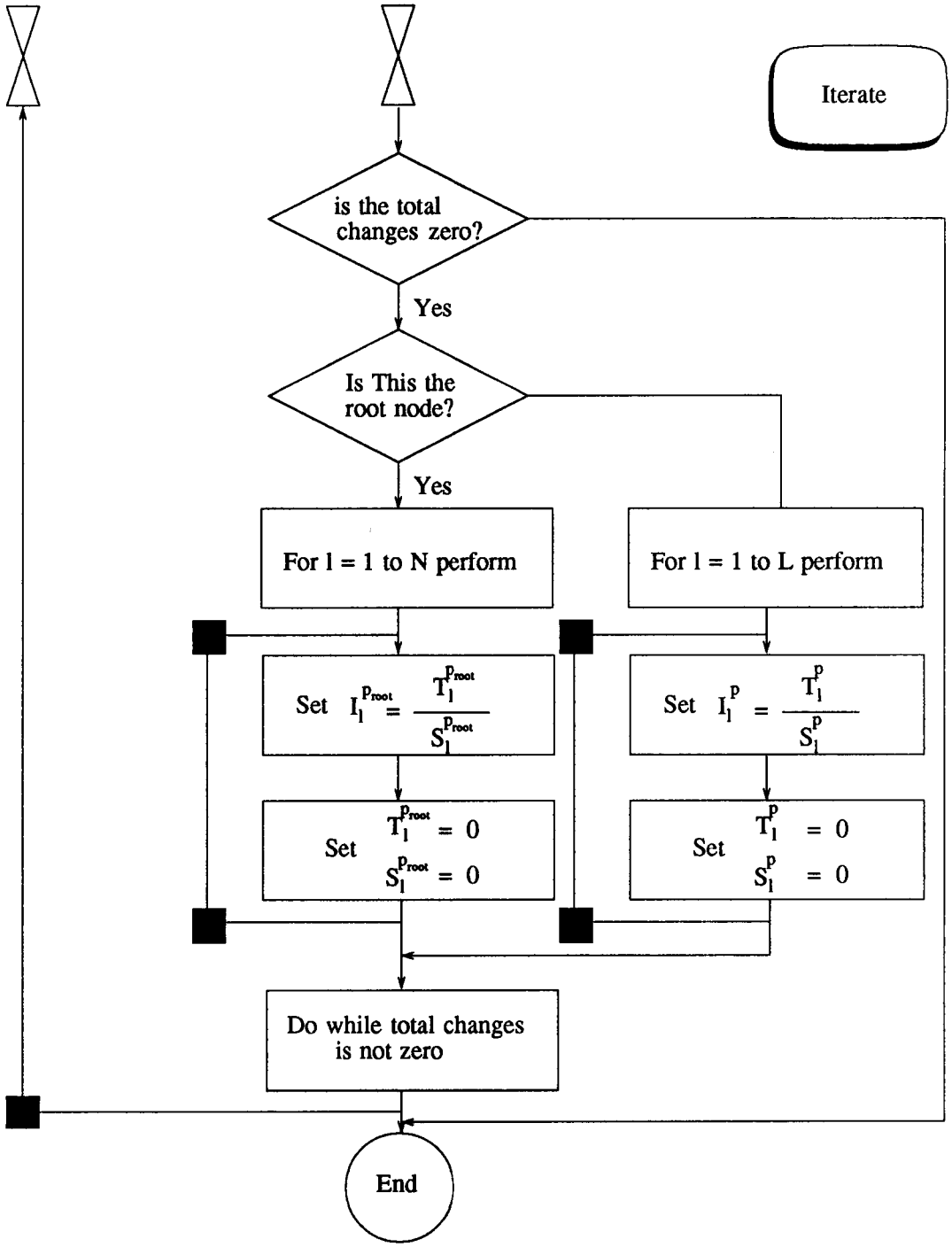

Iterate
Send buffer to node (P-1)
Receive buffer from node (P-1)
For l = 1 to L perform
Unload from buffer to row 2 of $I_l^p$
Does node (P+1) exist
No
Yes
For l = 1 to L perform
Buffer up row (last-1) of $I_l^p$


Iterate
Send buffer to node (P+1)
Receive buffer from node (P+1)
For l = 1 to L perform
Unload from buffer to last row of $I_1^p$
Is this the root node ?
No
Yes
Send $I_L$ to all nodes
Receive $I_L$ from root node
For l = 1 to L perform

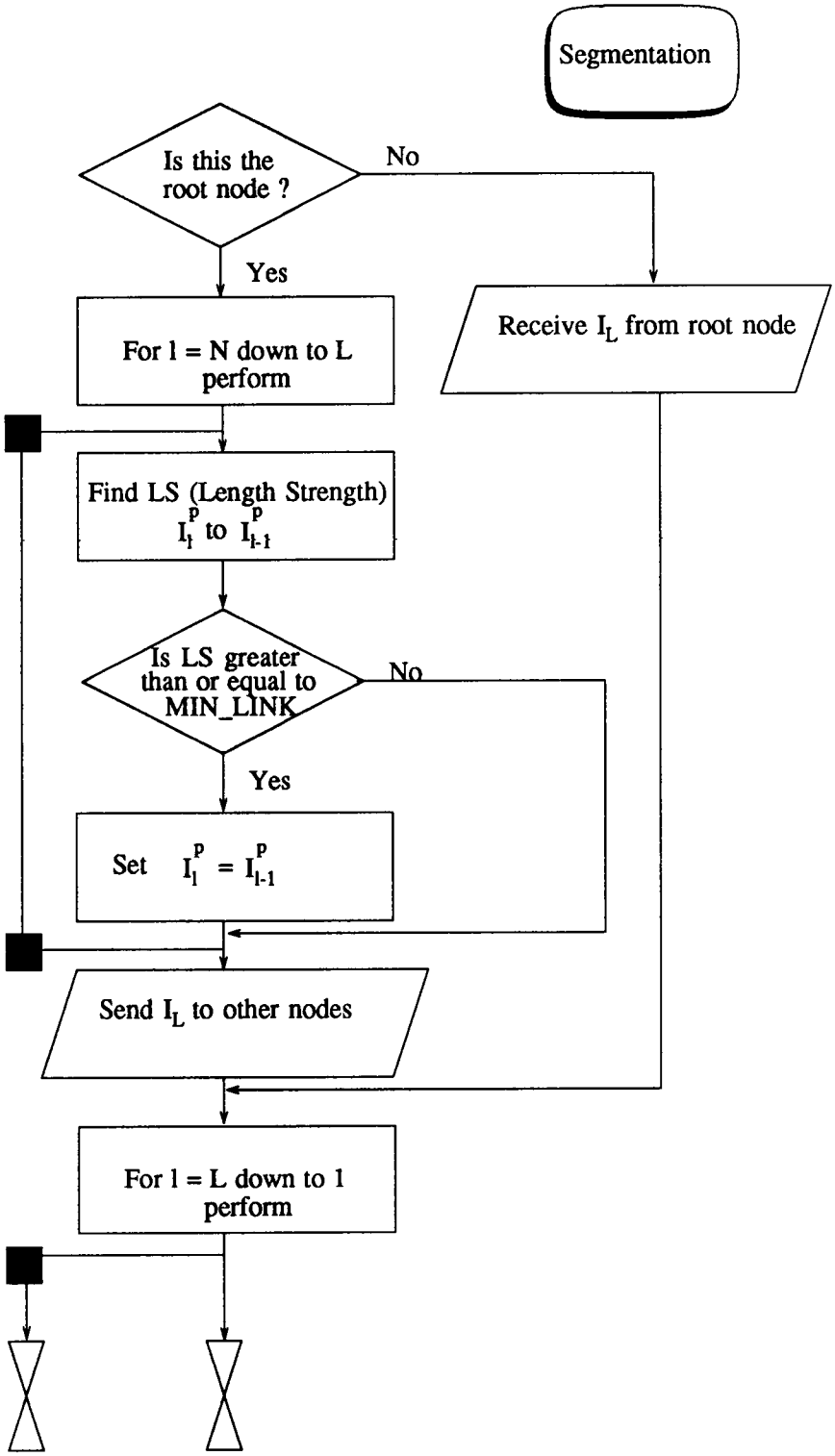

Iterate
Does node (P-1) exist?
Yes
No
Set start_row = 2
Set start_row = 1
Perform fuzzy linking on top left and right corner, and first row of $I_l^p$
Does node (P+1) exist?
Yes
No
Decrement last_row
Perform fuzzy linking on bottom left and right corner, and last row of $I_l^p$
For i = start_row to last_row perform

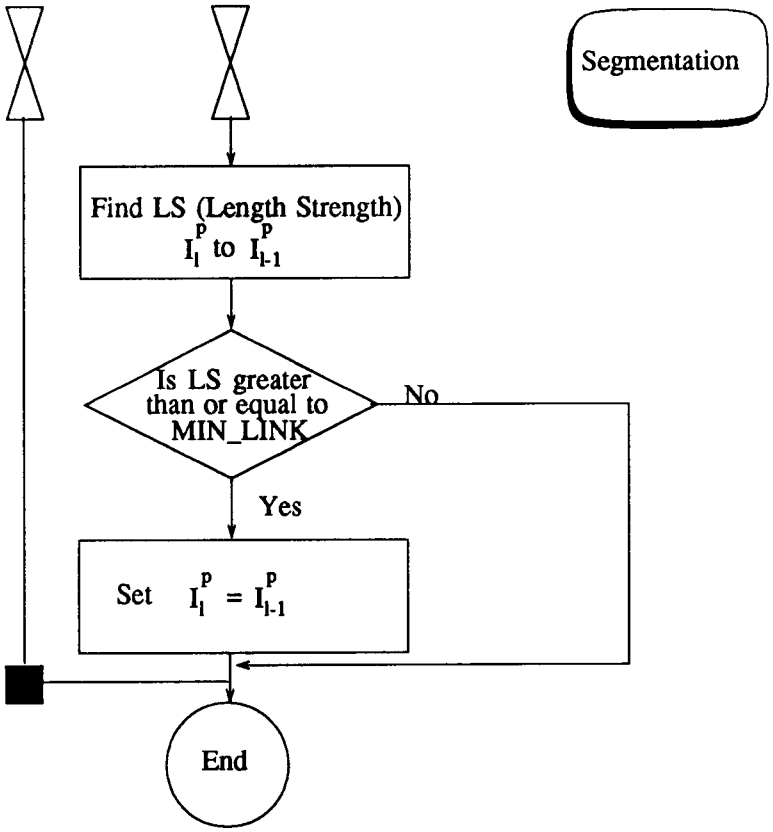

Iterate
Perform fuzzy linking on row (i) of $I_l^p$
Is (l+1) less than L?
No
Yes
Does node (P-1) exist?
No
Yes
Send row 1 of $S_l^p$ and $T_l^p$ to node (P-1)
Receive message from node (p-1) and add to row 2 of $S_l^p$ and $T_l^p$
Does node (P+1) exist?
No
Yes
Send row (last_row -1) of $S_l^p$ and $T_l^p$ to node (P+1)


Iterate
Receive message from node (p+1) and add to last row of $S_l^p$ and $T_l^p$
Is this the root node?
No
Yes
Set sub_changes = 0
For l = L to N perform
Perform fuzzy linking on l
Add sub_changes to changes $^{p_{root}}$
Find total changes by gadd()

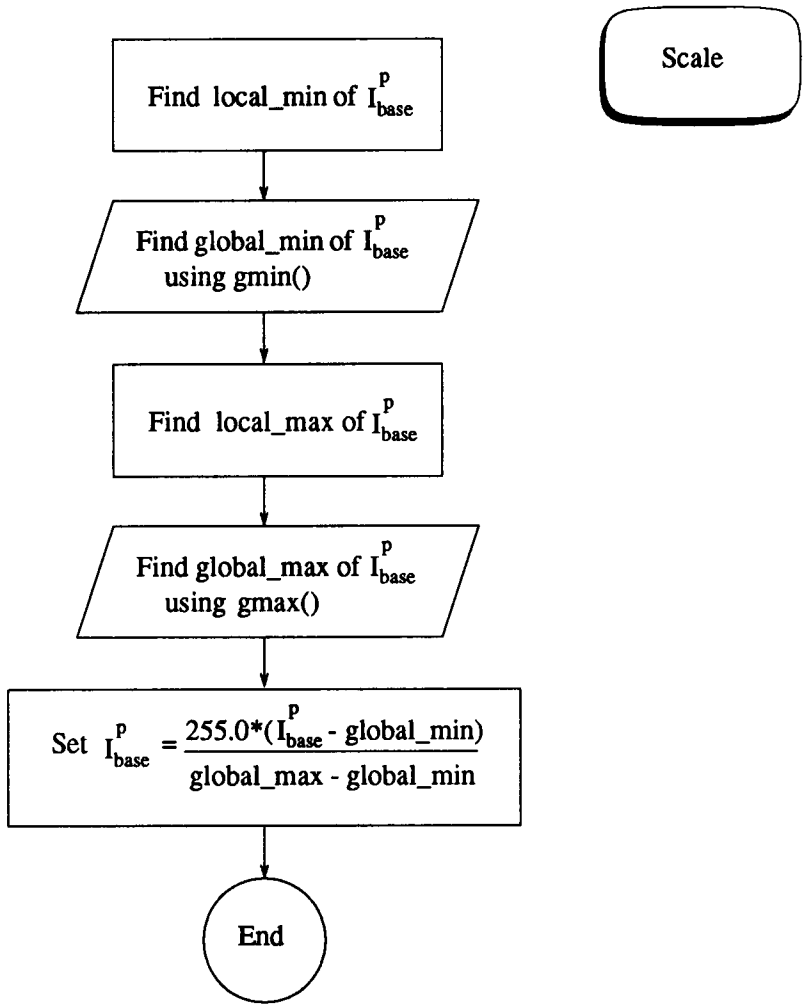

Iterate
is the total changes zero?
Yes
Is This the root node?
Yes
For l = 1 to N perform
For l = 1 to L perform
Set $I_l^{P_{root}} = \frac{T_l^{P_{root}}}{S_l^{P_{root}}}$
Set $I_l^{P} = \frac{T_l^{P}}{S_l^{P}}$
Set $T_l^{P_{root}} = 0$, $S_l^{P_{root}} = 0$
Set $T_l^{P} = 0$, $S_l^{P} = 0$
Do while total changes is not zero
End

### A.2.4 Performing Segmentation

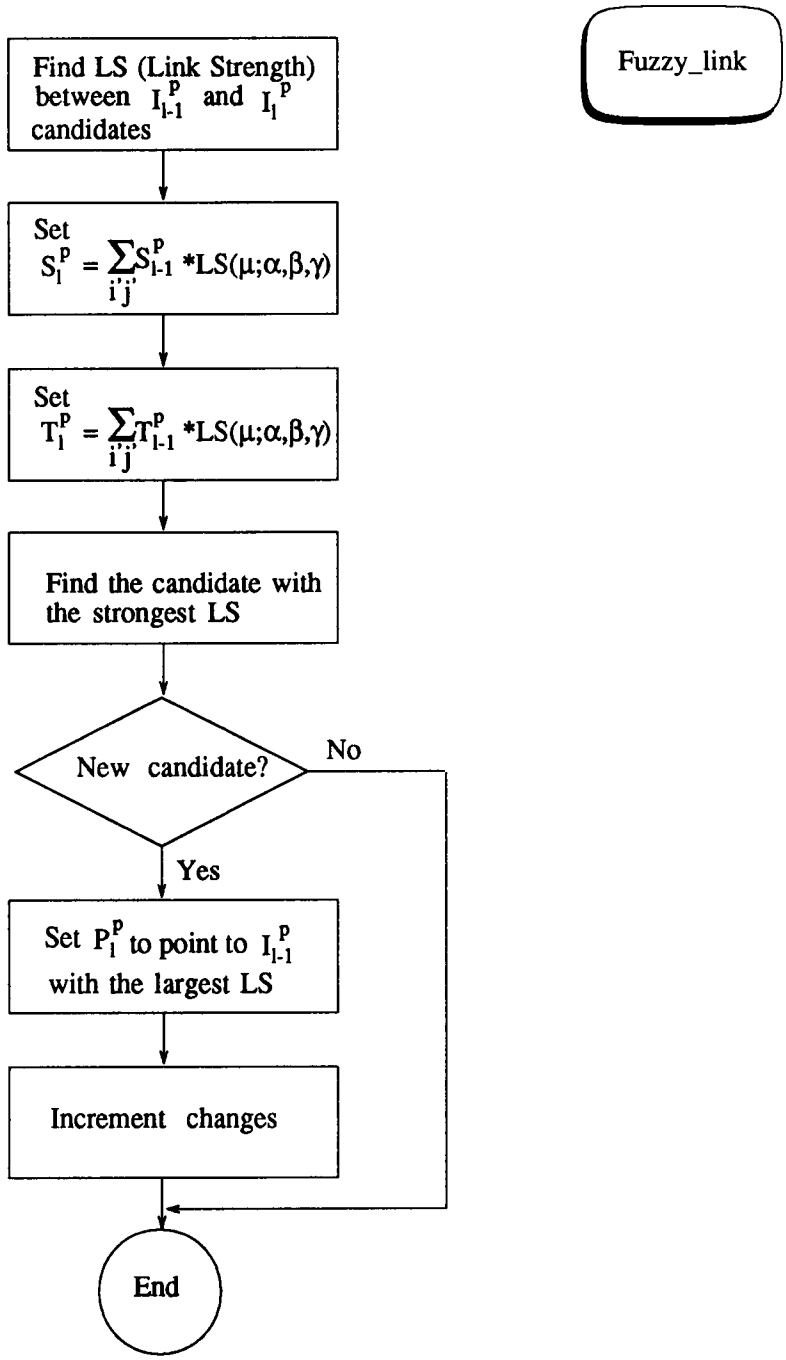



Segmentation
Find LS (Length Strength)
$I_l^p$ to $I_{l-1}^p$
Is LS greater than or equal to MIN_LINK
No
Yes
Set $I_l^p = I_{l-1}^p$
End

### A.2.5 Scaling the Result Image



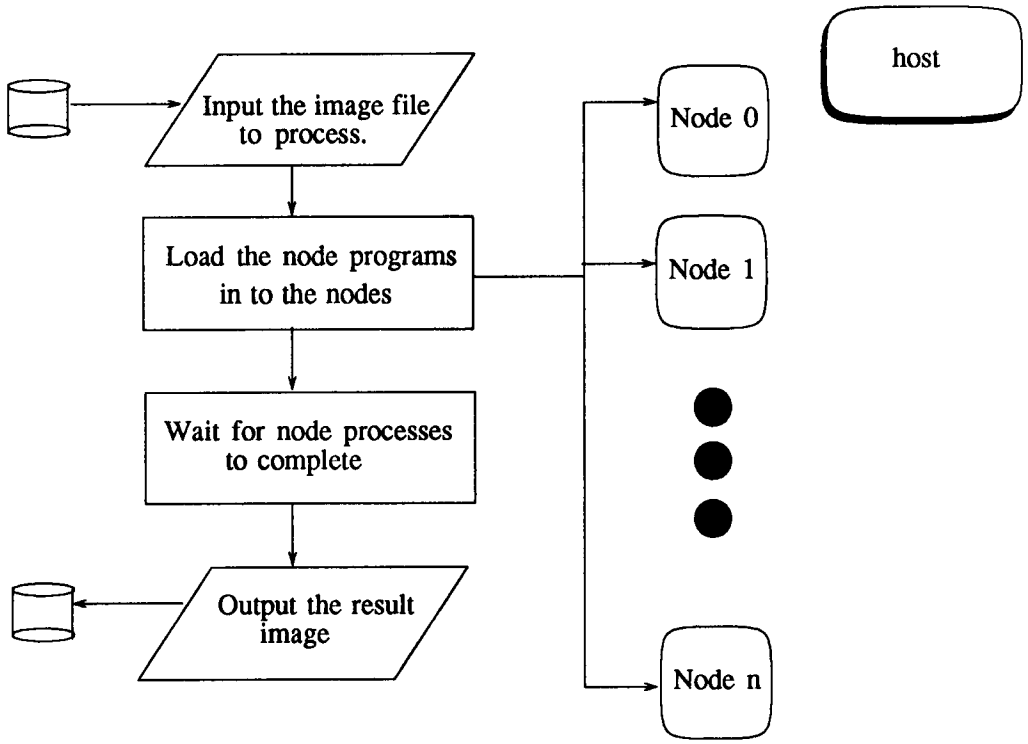
### A.2.6 Fuzzy Linking



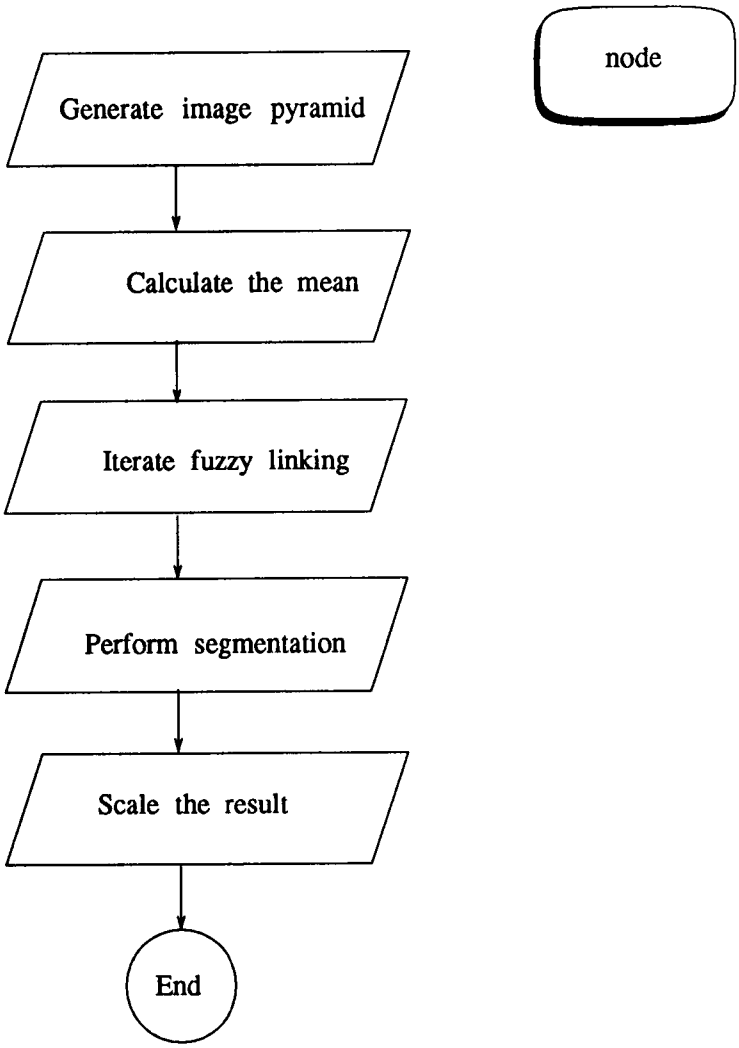
# APPENDIX B

# PYRAMID SEGMENTATION FOR SHARED MEMORY SYSTEMS

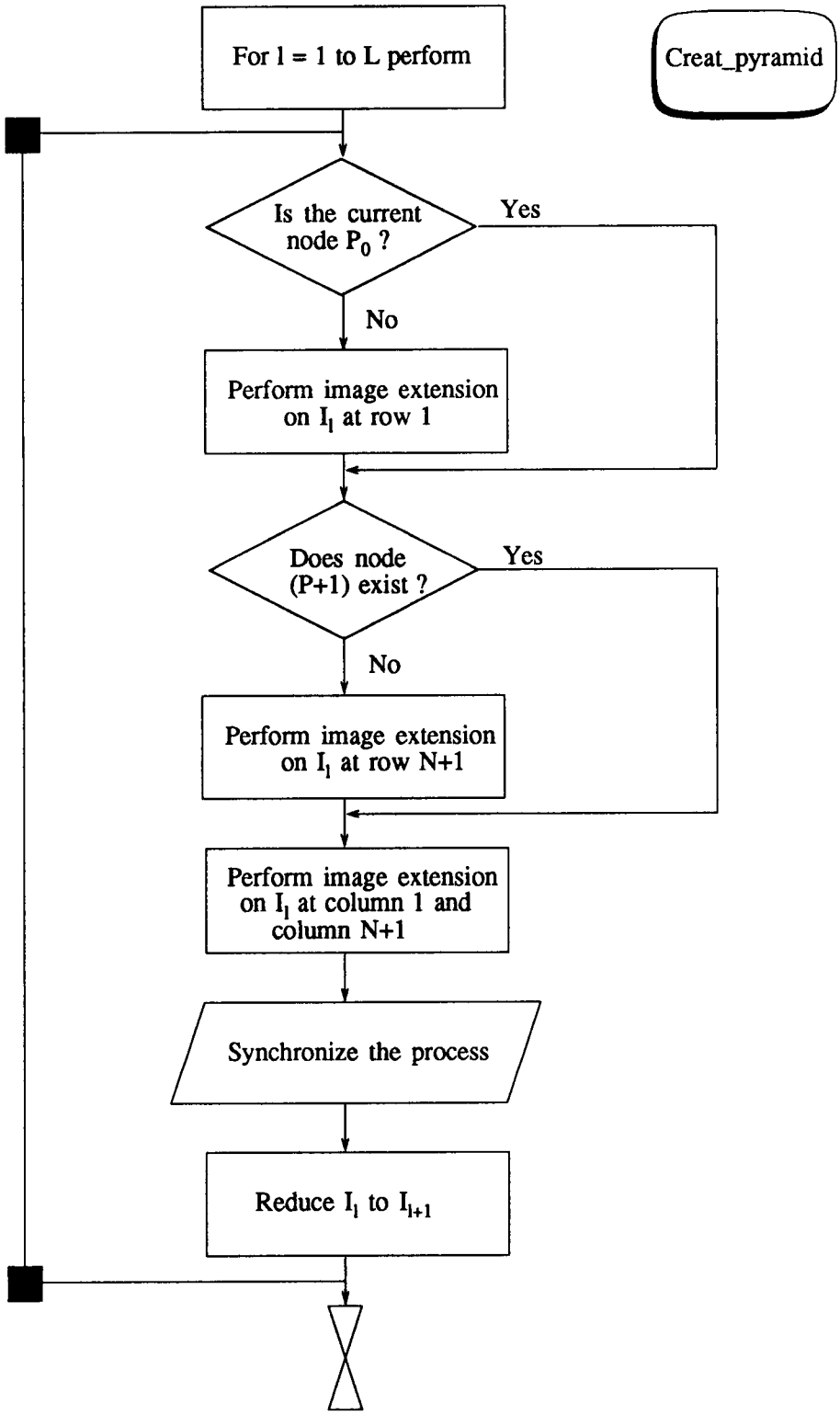
## B.1 Host Program

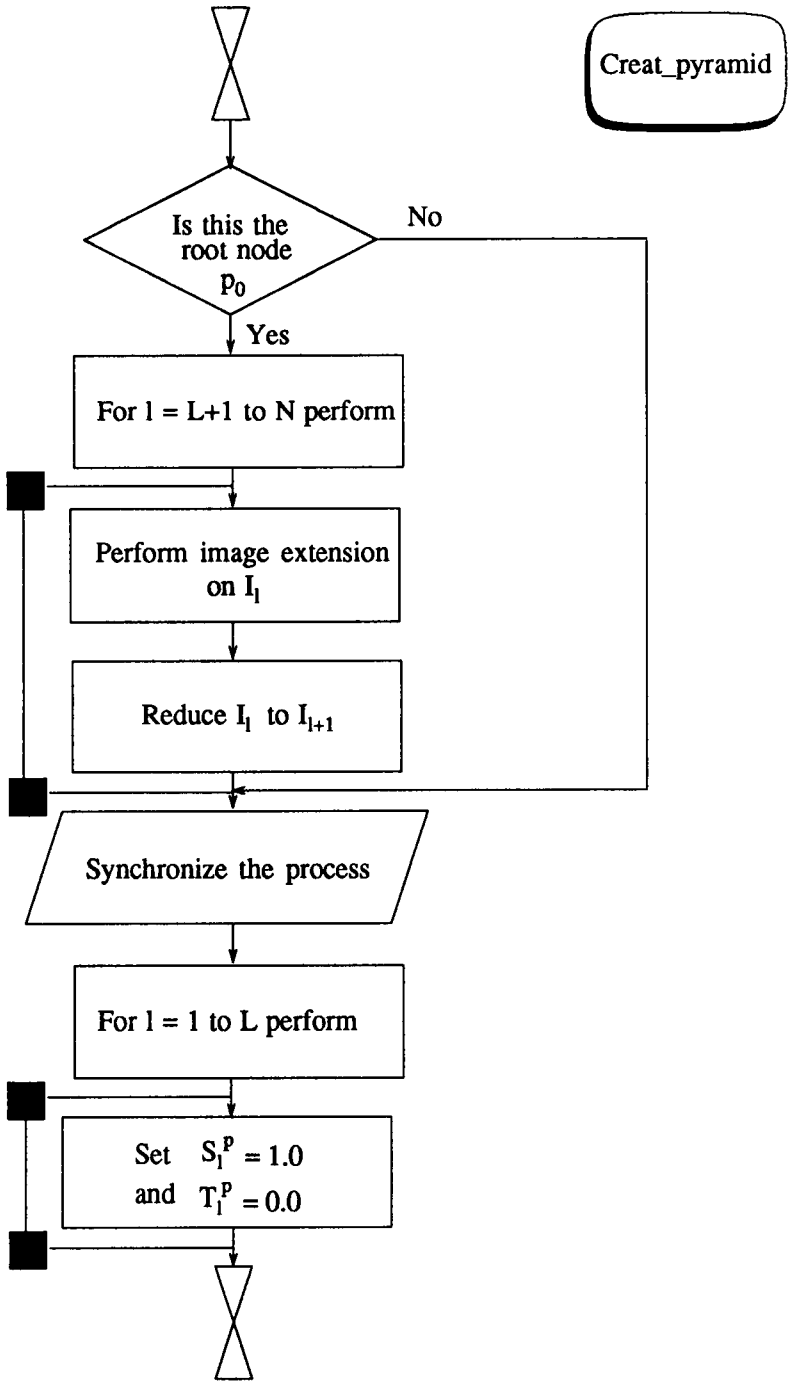


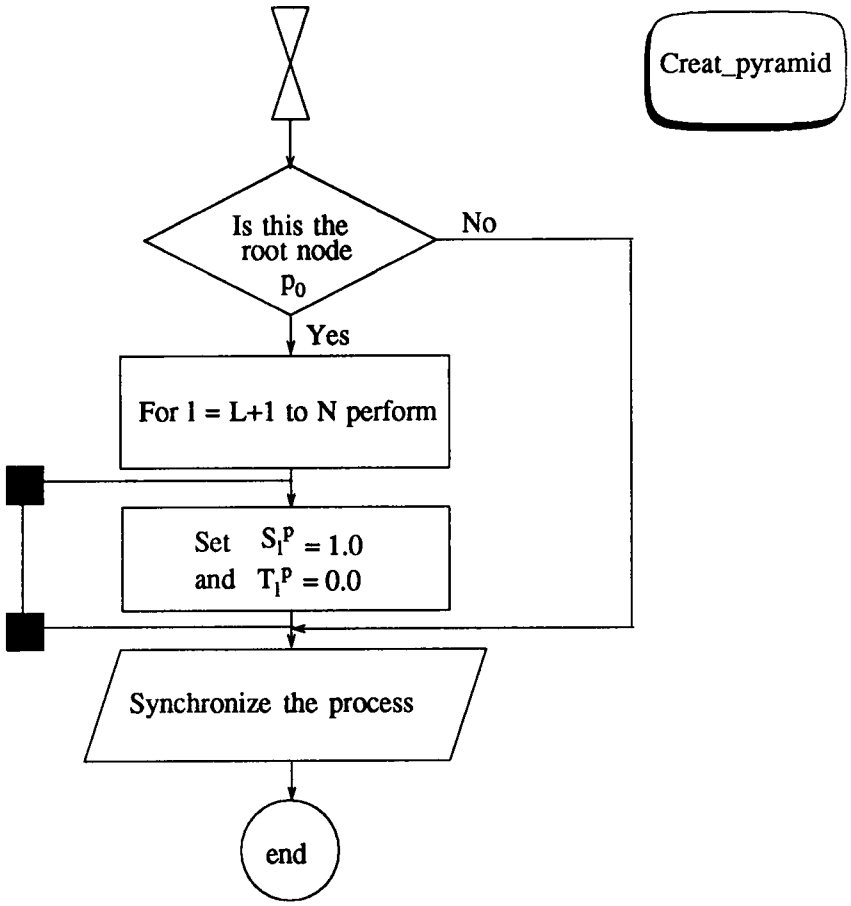
## B.2 Node Program



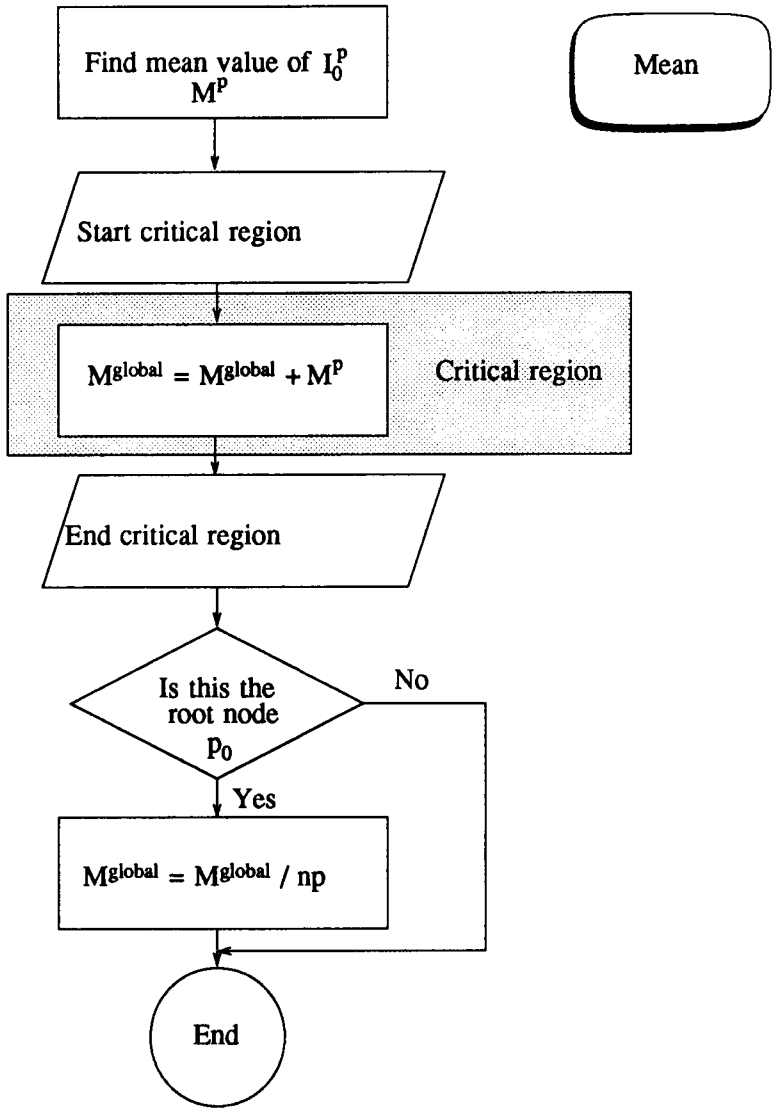
### B.2.1 Pyramid Creation



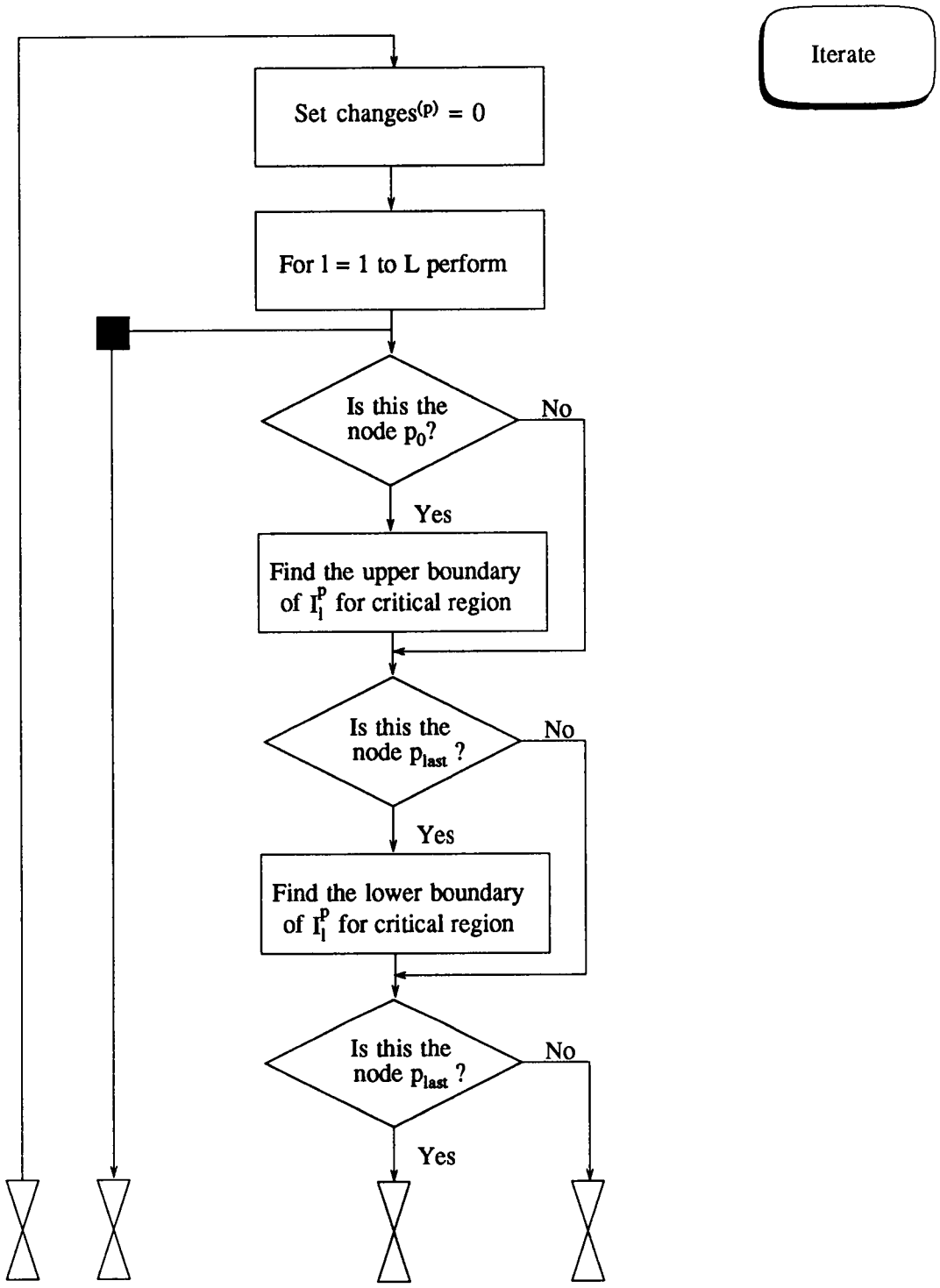

Creat_pyramid
Is this the root node $p_0$
No
Yes
For l = L+1 to N perform
Perform image extension on $I_l$
Reduce $I_l$ to $I_{l+1}$
Synchronize the process
For l = 1 to L perform
Set $S_l^p = 1.0$ and $T_l^p = 0.0$

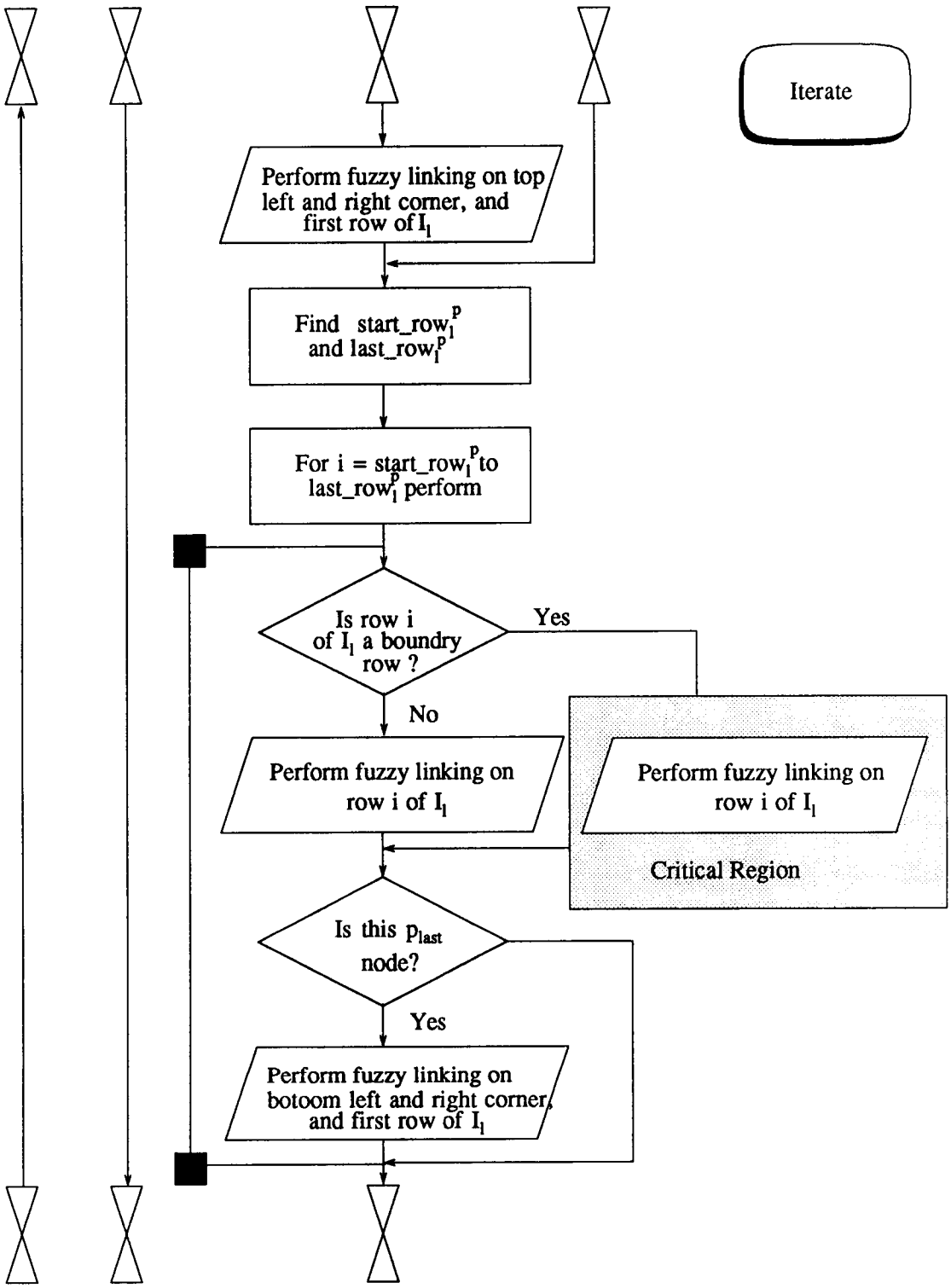

Creat_pyramid
Is this the root node $p_0$
No
Yes
For l = L+1 to N perform
Set $S_l^P = 1.0$ and $T_l^P = 0.0$
Synchronize the process
end

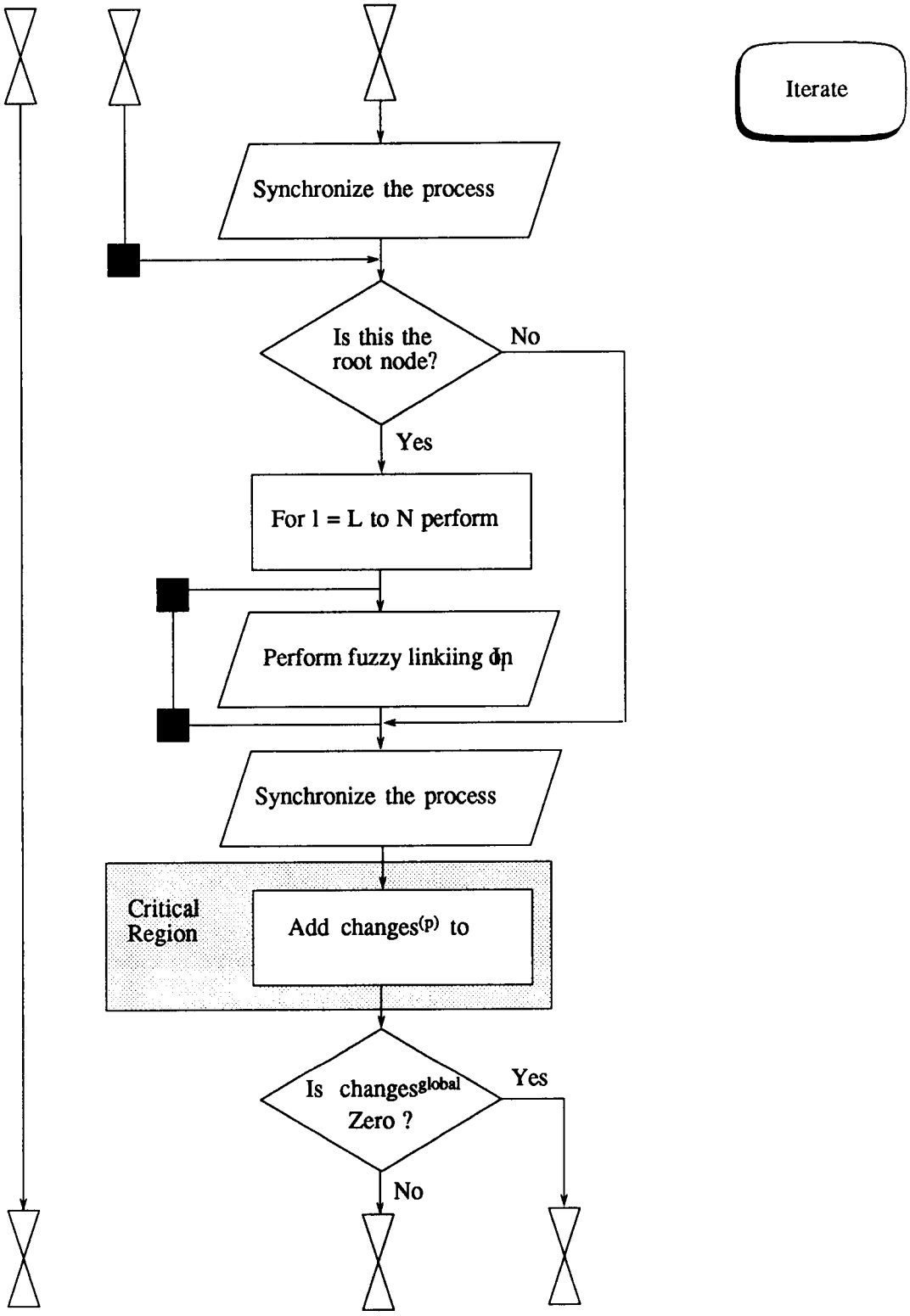
### B.2.2 Finding the Mean Value of Base of Pyramid

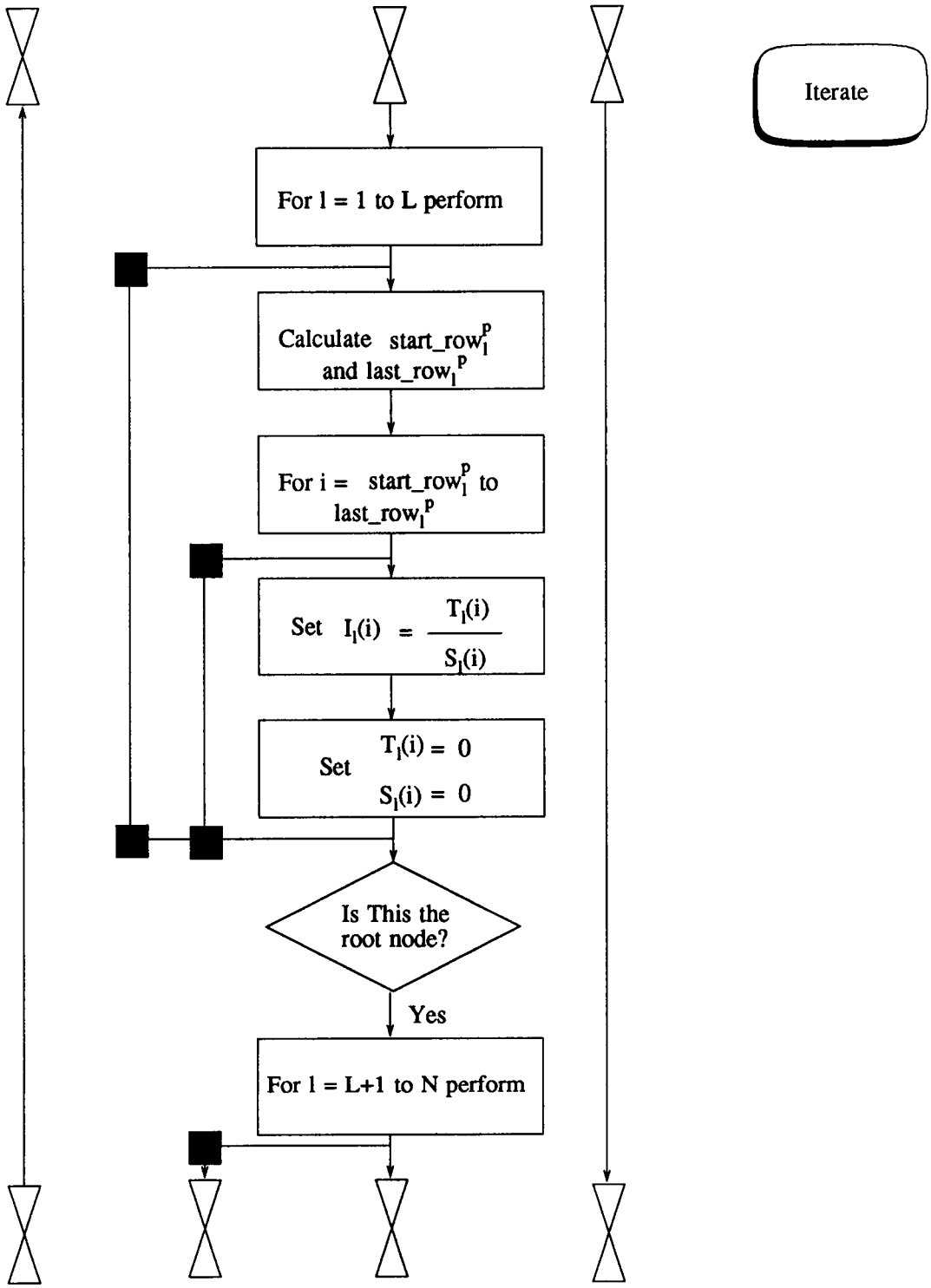


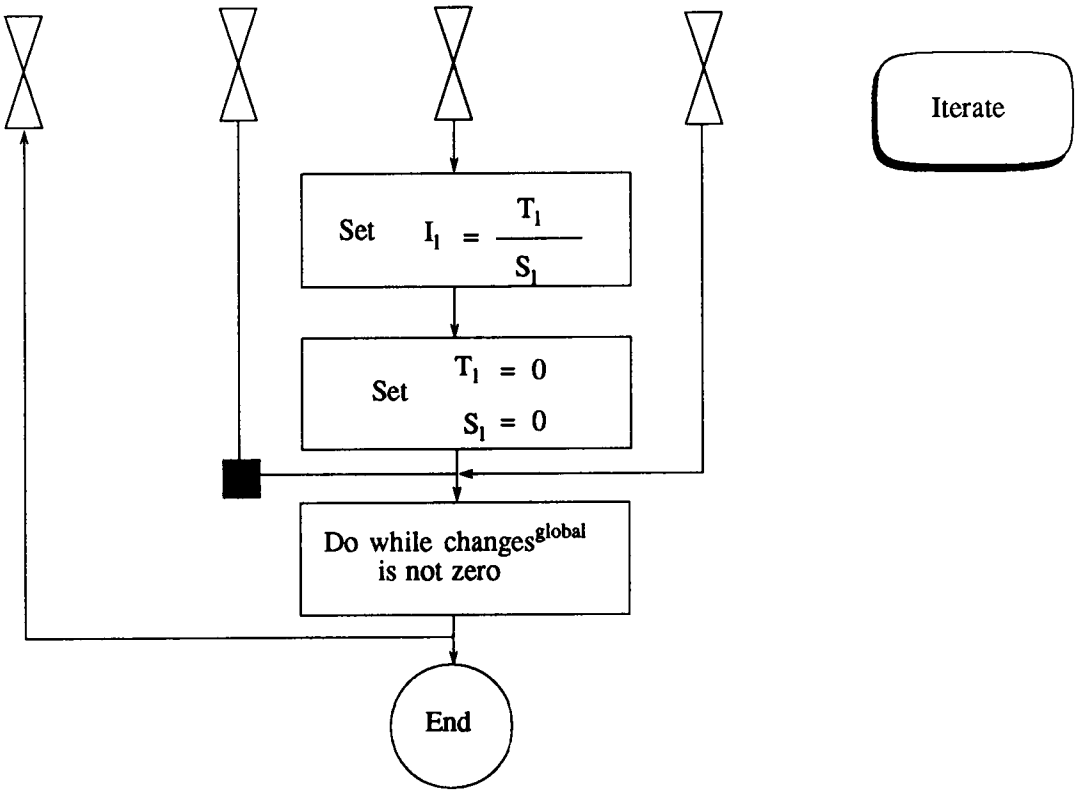
### B.2.3 Linking Process



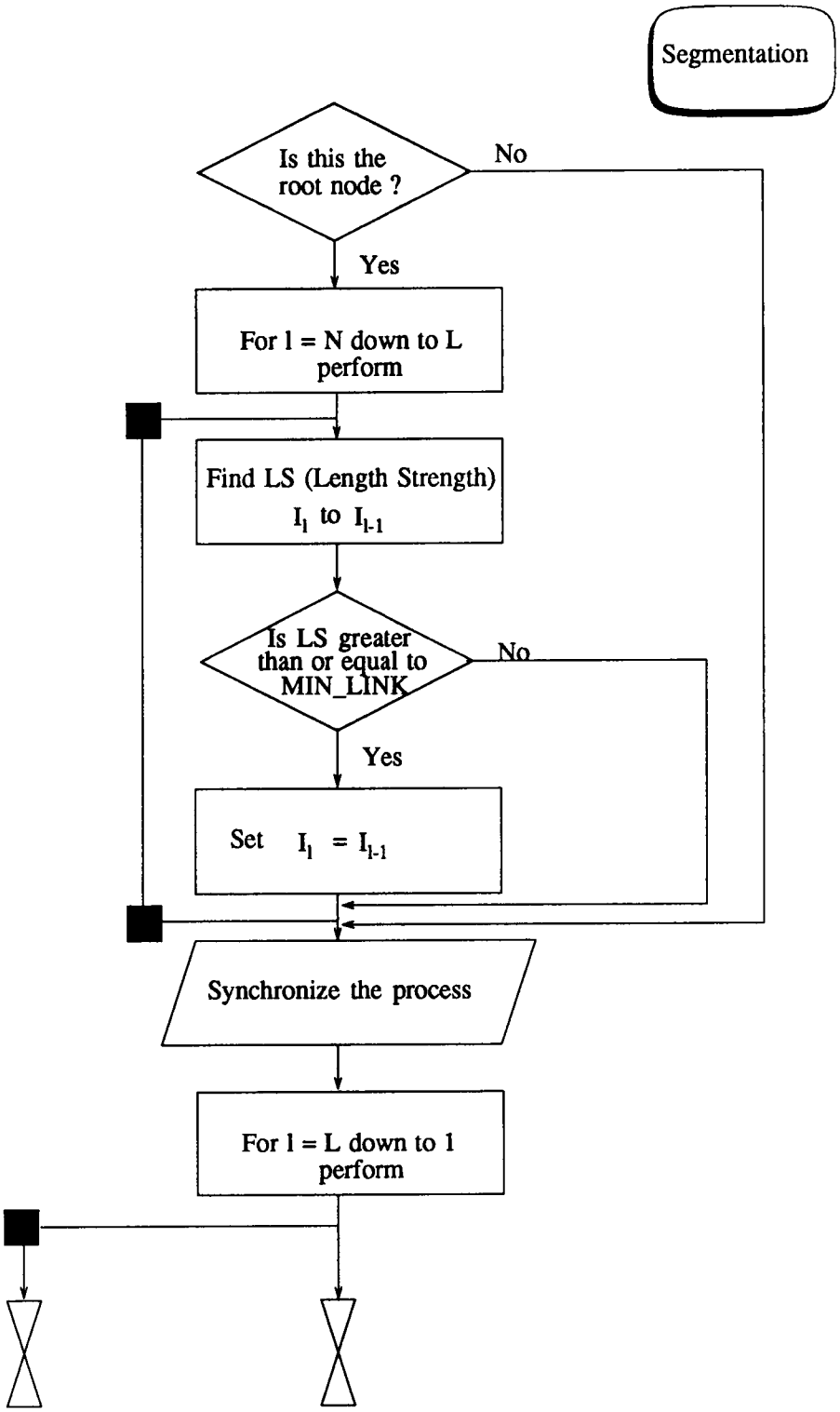

Iterate
Perform fuzzy linking on top left and right corner, and first row of $I_l$
Find $start_row_l^p$ and $last_row_l^p$
For i = $start_row_l^p$ to $last_row_l^p$ perform
Is row i of $I_l$ a boundry row ?
Yes
No
Perform fuzzy linking on row i of $I_l$
Perform fuzzy linking on row i of $I_l$
Critical Region
Is this $p_{last}$ node?
Yes
Perform fuzzy linking on botoom left and right corner, and first row of $I_l$

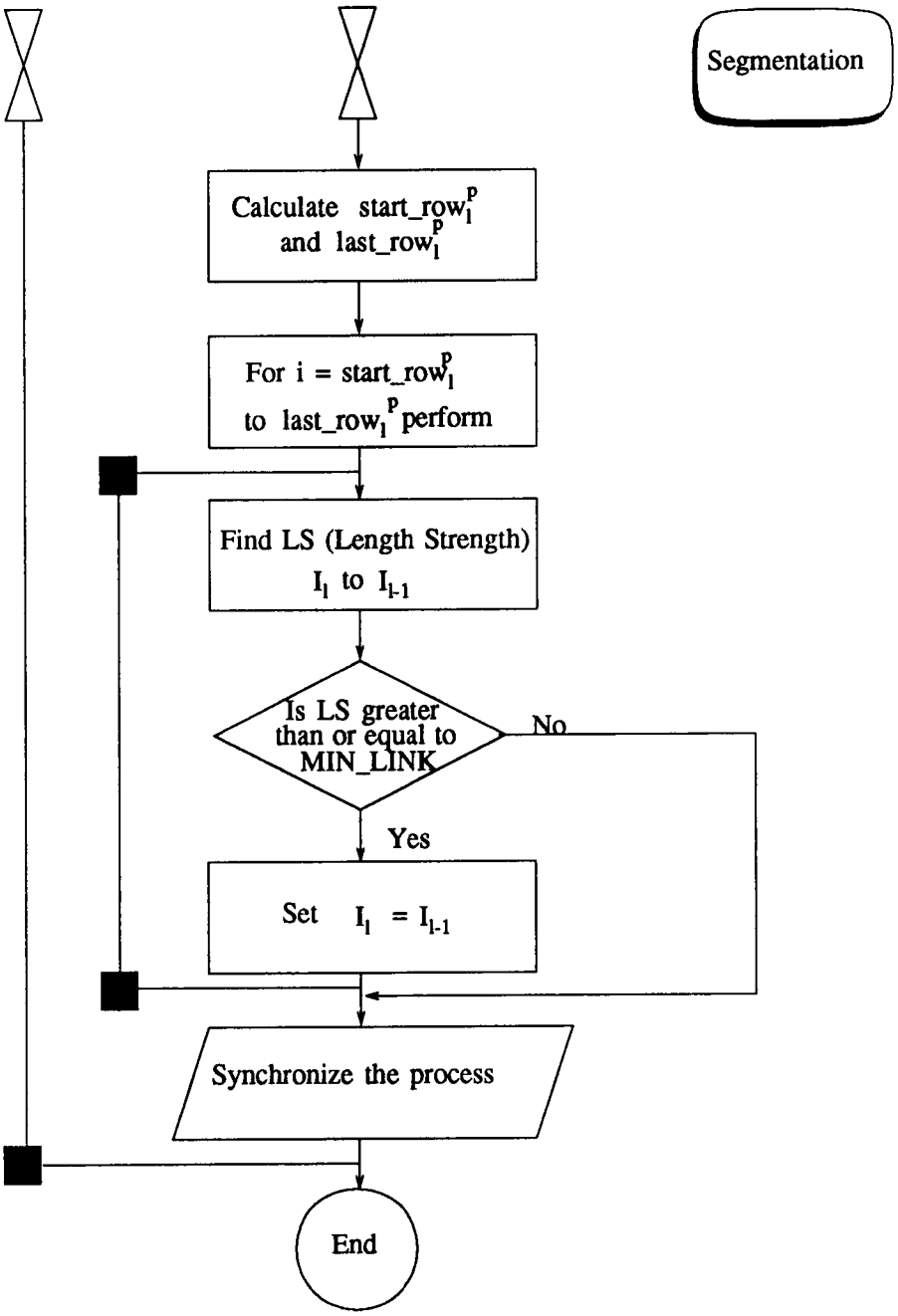

Iterate
Synchronize the process
Is this the root node?
No
Yes
For l = L to N perform
Perform fuzzy linkiing on
Synchronize the process
Critical Region
Add changes(p) to
Is changesglobal Zero ?
Yes
No


Iterate
For l = 1 to L perform
Calculate start_row$_l^p$ and last_row$_l^p$
For i = start_row$_l^p$ to last_row$_l^p$
Set $I_l(i) = \frac{T_l(i)}{S_l(i)}$
Set $T_l(i) = 0$ $S_l(i) = 0$
Is This the root node?
Yes
For l = L+1 to N perform

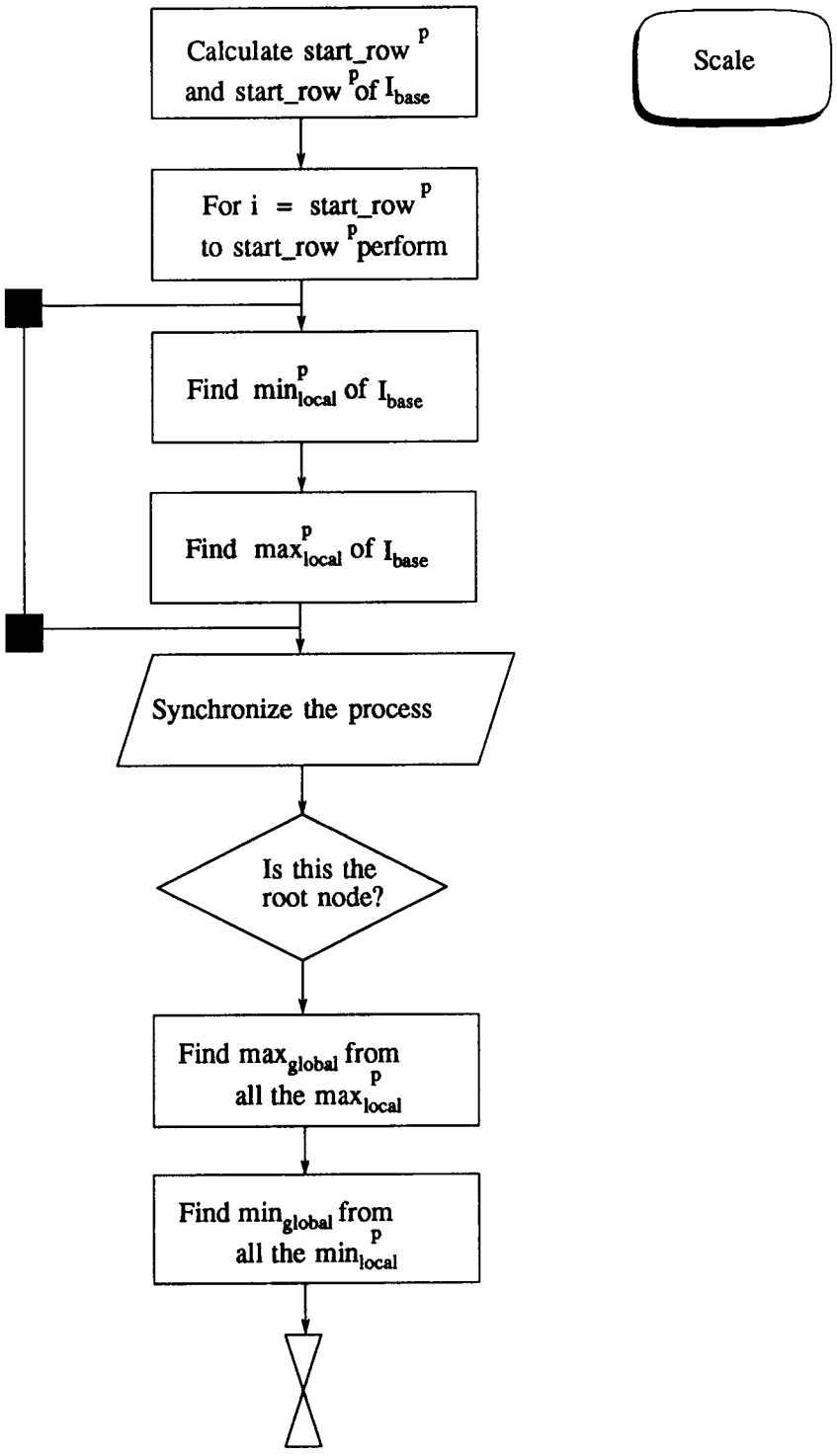

Iterate
Set $I_l = \frac{T_l}{S_l}$
Set $T_l = 0$ $S_l = 0$
Do while changes$^{global}$ is not zero
End

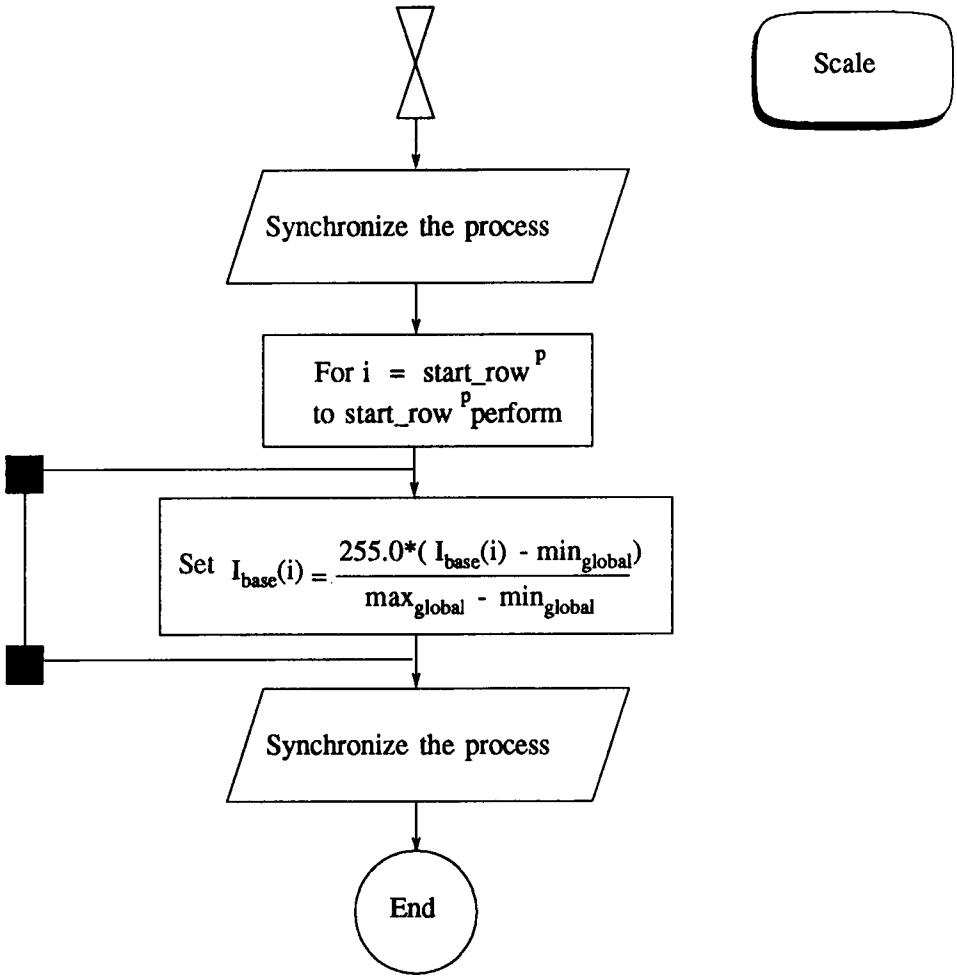
### B.2.4 Performing Segmentation



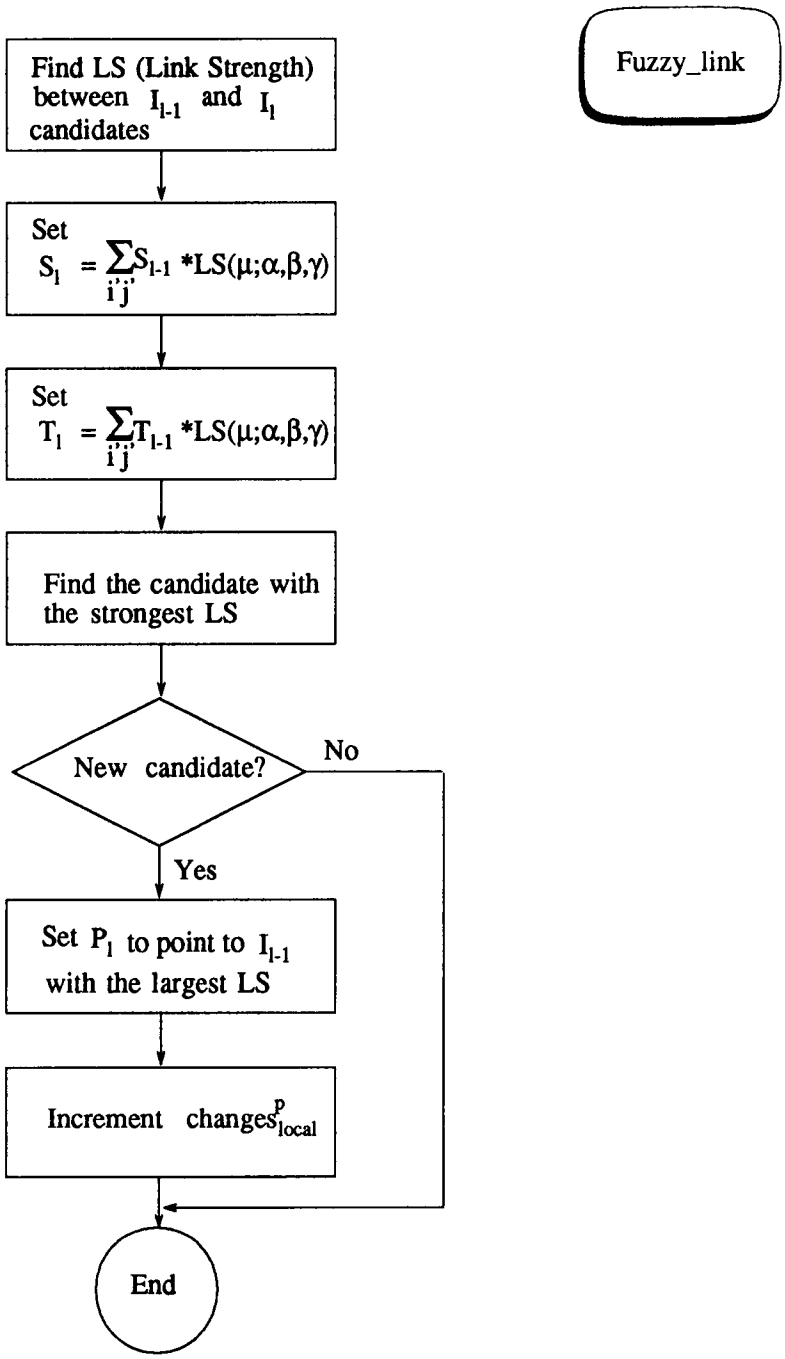

Segmentation
Calculate start_row$_l^p$ and last_row$_l^p$
For i = start_row$_l^p$ to last_row$_l^p$ perform
Find LS (Length Strength) $I_l$ to $I_{l-1}$
Is LS greater than or equal to MIN_LINK
No
Yes
Set $I_l = I_{l-1}$
Synchronize the process
End

### B.2.5 Scaling the Result Image




Scale
Synchronize the process
For i = start_row p
to start_row p perform
Set $I_{base}(i) = \frac{255.0*(I_{base}(i) - min_{global})}{max_{global} - min_{global}}$
Synchronize the process
End

### B.2.6 Fuzzy Linking



# VITA

Mehrdad Azimi Saravi was born in Tehran, Iran on September 13, 1960. He was graduated from the Sokhan School in Tehran, Iran in August 1977. He entered the University of Tennessee in September 1983 and received his Bachelor of Science degree in Computer Science in March 1987. He was a research assistant starting in September 1985 and involved in system management starting in January 1986. He received the Master of Science degree in Engineering Science and Mechanics in May 1992.