AD-A197 341

Unclassified

DTIC FILE COPY

4

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

| REPORT DOCUMENTATION PAGE | | READ INSTRUCTIONS BEFORE COMPLETING FORM |
|---|---|---|
| 1. REPORT NUMBER | 2. GOVT ACCESSION NO. | 3. RECIPIENT'S CATALOG NUMBER |
| 4. TITLE (and Subtitle) Practical Algorithms for Image Component Labeling on SIMD Mesh Connected Computers | | 5. TYPE OF REPORT & PERIOD COVERED |
| | | 6. PERFORMING ORG. REPORT NUMBER TR-87-12-05 |
| 7. AUTHOR(s) R. E. Cypher and L. Snyder | | 8. CONTRACT OR GRANT NUMBER(s) N00014-86-K-0264 |
| 9. PERFORMING ORGANIZATION NAME AND ADDRESS University of Washington Department of Computer Science Seattle, Washington 98195 | | 10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS |
| 11. CONTROLLING OFFICE NAME AND ADDRESS Office of Naval Research Information Systems Program Arlington, VA 22217 | | 12. REPORT DATE December 1987 |
| | | 13. NUMBER OF PAGES 8 |
| 14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) | | 15. SECURITY CLASS. (of this report) Unclassified |
| | | 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE |

16. DISTRIBUTION STATEMENT (of this Report)

Distribution of this report is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

parallel algorithms, image processing, mesh, connected component labeling

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Two new parallel algorithms are presented for the problem of labeling the connected components of a binary image, which is also known as the "connected ones problem". The machine model is an SIMD two-dimensional mesh connected computer consisting of an N x N array of processing elements, each containing a single pixel of an N x N image. Both new algorithms use a "shrinking" operation defined by Levialdi and have time complexities of O(N log N) bit operations, which makes them the fastest local algorithms for the problem. Compared with other

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

approaches having similar or better time complexities, this local approach dramatically simplifies the algorithms and reduces the constants of proportionality by nearly two orders of magnitude, thus making them the first practical algorithms for the problem. The two algorithms differ in the amount of memory required per processing element; the first uses O(N) bits while the second employs a novel compression scheme to reduce the requirement to O(log N) bits.

# Practical Algorithms for Image Component Labeling on SIMD Mesh Connected Computers

R.E. Cypher and L. Snyder
Department of Computer Science, FR-35
University of Washington
Seattle, Washington 98195

J. L. C. Sanz
Computer Science Department
IBM Almaden Research Center
San Jose, California 95120

TR 87-12-05
July 1987

Two new parallel algorithms are presented for the problem of labeling the connected components of a binary image, which is also known as the "connected ones problem". The machine model is an SIMD two-dimensional mesh connected computer consisting of an N × N array of processing elements, each containing a single pixel of an N × N image. Both new algorithms use a "shrinking" operation defined by Levialdi and have time complexities of O(N log N) bit operations, which makes them the fastest local algorithms for the problem. Compared with other approaches having similar or better time complexities, this local approach dramatically simplifies the algorithms and reduces the constants of proportionality by nearly two orders of magnitude. thus making them the first practical algorithms for the problem. The two algorithms differ in the amount of memory required per processing element; the first uses O(N) bits while the second employs a novel compression scheme to reduce the requirement to O(log N) bits.

# Practical Algorithms for Image Component Labeling on SIMD Mesh Connected Computers (Preliminary Version)

R.E. Cypher*, J.L.C. Sanz,** and L. Snyder*

Abstract

Two new parallel algorithms are presented for the problem of labeling the connected components of a binary image, which is also known as the "connected ones problem." The machine model is an SIMD two-dimensional mesh connected computer consisting of an N × N array of processing elements, each containing a single pixel of an N × N image. Both new algorithms use a "shrinking" operation defined by Levialdi and have time complexities of O(N log N) bit operations, which makes them the fastest local algorithms for the problem. Compared with other approaches having similar or better time complexities, this local approach dramatically simplifies the algorithms and reduces the constants of proportionality by nearly two orders of magnitude, thus making them the first practical algorithms for the problem. The two algorithms differ in the amount of memory required per processing element; the first uses O(N) bits while the second employs a novel compression scheme to reduce the requirement to O(log N) bits.

## 1 Introduction

The tasks encountered in machine vision can be roughly divided into three classes based on the data structures they use. Low level tasks operate on large 2-dimensional arrays of pixels. High level tasks operate on smaller symbolic data structures such as graphs that are intended to describe the scene under analysis in a manner closer to human understanding. Intermediate level tasks link the low and high levels by taking an array of pixels as input and creating a symbolic data structure as output. Creating parallel architectures for intermediate level vision tasks is particularly difficult because both symbolic and iconic (pixel array) data structures must be accommodated. The design of parallel architectures for the various image processing task levels is currently a topic of great interest to the machine vision community [1-4].

This paper addresses an important intermediate level task in machine vision: labeling the connected components of a binary image. New algorithms are presented which run on an SIMD mesh connected computer consisting of an N × N array of processing elements, each of which holds a single pixel of an N × N image. The problem of image component labeling, also known as the "connected ones problem", consists of associating labels with the 1 valued pixels of a binary image such that any two pixels have the same label if and only if they lie in the same connected component, where a connected component is a maximal region of 1 valued pixels such that any two pixels in the region lie on a path that is connected and only passes through pixels with value 1. The two common definitions of connectedness are 4-connectedness and 8-connectedness. Two pixels are 4-connected if they are adjacent vertically or horizontally, and they are 8-connected if they are adjacent vertically, horizontally or diagonally [5]. The labeling of connected components has been intensively studied [6-10] and is important in many applications. It allows regions (the connected components) to be identified so that the analysis of the image can be performed on a higher level than the pixel level.

A two-dimensional mesh connected computer consists of a large number of processing elements (PEs) arranged in a square array, as shown in Figure 1. Each PE consists of a processor and an associated memory. For the number of PEs in the array to approach the number of pixels in a typical image (for example, $2^{18}$), the PEs must be simple and inexpensive. In particular, the PEs considered here are bit serial machines that operate in a Single Instruction Stream, Multiple Data Stream (SIMD) mode, with all control signals coming from a single control unit. The control unit reads instructions from its private memory, decodes them, and broadcasts the control signals to the PE array. In addition to broadcasting the control information to the processors, the control unit sends addresses to the memory units, so every PE accesses the same memory location at a given time.

Each PE has a special register called a mask register. When an instruction is sent from the controller to the array of PEs, only those PEs with a 1 in their mask register perform the instruction; all others do nothing. This allows operations to be performed on a subset of the PEs in a data dependent manner. Of course, there are some instructions which operate on all PEs regardless of the setting of the mask registers, thus allowing the disabled PEs to be used again.

The two-dimensional mesh interconnection structure is easy to construct because it is regular, it has short connections, it requires only 4 connections per PE, and it is possible to build in two dimensions without having any connections cross. The bit serial processors have a one bit wide data path to their four nearest neighbors. Commercial versions of such machines include CLIP4 [11], the MPP [12] and the GAPP [13].

*Department of Computer Science, FR-35, University of Washington, Seattle, Washington 98195

**Computer Science Department, IBM Almaden Research Center, San Jose, California 95120

This paper presents two new algorithms for labeling the connected components of an image on mesh connected computers, together with comparisons with previously published algorithms. The authors believe these are the first practical algorithms for labeling connected components on large mesh connected computers, because they have very modest architectural requirements and are nearly two orders of magnitude faster than previously published algorithms.

## 2 Labeling Connected Components

A variety of algorithms are known for connected component labeling, so it is convenient to divide them into two classes: *local algorithms* and *random access algorithms*. Local algorithms repeatedly change the contents of each processor based on the contents of neighboring processors. As will be explained in the next section, all previously published local algorithms require O($N^2$ log N) bit operations, while the local algorithms presented here need only O(N log N) bit operations. Random access algorithms achieve good asymptotic performance by using complex pointer manipulation routines. For example, Nassimi and Sahni [9] give an algorithm requiring O(N log N) bit operations and O(log N) bits of memory, matching the bounds of the second algorithm presented here. Interestingly, though the bit operations measure of complexity seems preferable, word operations are also used as a measure; adopting this measure, both Nassimi and Sahni's algorithm and the first algorithm presented here require O(N) time, though the algorithm given here also requires O(N) bits of memory per processor.

The importance of the new local algorithms presented here is not that they match or nearly match the best known asymptotic complexity, but that their constants of proportionality are very small. The complexity of random access algorithms causes their constants to be large. For example, the Nassimi and Sahni algorithm can be shown to require 1276 N log N + $\theta(N^{1/2}$ log N) communication operations, while the two algorithms reported here require 12N log N + 4N and 14N log N communications operations, respectively, in the 4-connected case. (Similar results apply for the 8-connected case.) It seems likely that Nassimi and Sahni's algorithm can be modified, using techniques developed by Stout [17], to yield an O(N) time algorithm. However, it is likely that such an algorithm would also have a large constant of proportionality, making it inferior to those presented here for practical values of N.

### 2.1 Local Neighborhood Algorithms

A well known local neighborhood algorithm for labeling the connected components of an image, which will be called the "component broadcasting algorithm", has a worst case time complexity of O($N^2$ log N) bit operations. Each PE containing a pixel with value 1 is initially assigned a label that is the concatenation of its x and y coordinates. Then a series of broadcasting operations is performed. A broadcasting operation consists of transferring the label of each PE with a 1-valued pixel to each of its (4- or 8-connected) neighbor PEs having a 1-valued pixel. Then every PE calculates the minimum of its current label and the labels which it has received, taking this minimum as its new label. The connected components are correctly labeled when a broadcasting operation fails to change any of the labels. Each broadcasting operation requires O(log N) bit operations, and it will be shown that O($N^2$) broadcasting operations may be required.

Another O($N^2$ log N) time local neighborhood algorithm is given by C. Dyer and A. Rosenfeld [14]. It consists of first identifying a special pixel in each component and assigning a unique label to each of the special pixels. The special pixels are identified by using an algorithm developed by S. Kosaraju [15]. The next step consists of building a minimum spanning tree for each component that is rooted at a special pixel. The labels are then broadcast from the special pixels to the other PEs in the component using the spanning trees. This operation is very similar to the component broadcasting algorithm given above.

In order to analyze the time complexity of the above algorithms, it is useful to introduce two new terms. The "intrinsic distance" between two pixels in the same connected component is one less than the number of pixels in the shortest (4- or 8-connected) path between them composed only of pixels with value 1. The "intrinsic diameter" of a connected component is the largest intrinsic distance between any pair of pixels in the component. The number of broadcasting operations required by the above algorithms is proportional to the largest of the intrinsic diameters of the connected components in the image. In images with small, convex connected components, the intrinsic diameters are small and the algorithms provide the desired labeling quickly. But some images have very long and thin connected components with intrinsic diameters proportional to the $N^2$ area of the image. One such example is shown in Figure 2. When it is safe to assume that no connected components will have an intrinsic diameter greater than N, the above algorithms may be the best possible.

The first new algorithm given here performs connected components labeling in O(N log N) bit operations in the worst case using O(N) bits of memory. This algorithm, to be called the "component shrinking algorithm", is based on repeated application of a binary morphological operation defined by S. Levialdi [16]. The value of pixel P(i,j) is determined from the previous values of pixels P(i,j), P(i+1,j), P(i,j-1) and P(i+1, j-1). For 8-connectedness, the new value of pixel P(i,j) is defined to be

$$h(h(P(i,j) + P(i,j-1) + P(i+1,j) - 1) + h(P(i,j) + P(i+1,j-1) - 1))$$

where h(t) is the "Heaviside" function defined as follows: $h(t) = 0$ for $t \leq 0$, $h(t) = 1$ for $t > 0$. The $2 \times 2$ neighborhoods that create a 0 are shown in the upper part of Figure 3.

The effect of this operation, called the "shrinking operation", can be easily understood as follows. Assume that pixel P(i,j) is in row i and column j and that pixel P(0,0) is in the lower lefthand corner of the image. Then if pixel P(i,j) originally has value 1, it will have value 1 after the shrinking operation if and only if at least one of its three neighbors to the left, above, or diagonally left and above has a 1. If pixel P(i,j) originally has value 0, it will have value 1 after the shrinking operation if and only if both its neighbor to the left and its neighbor above have 1s. Levialdi proved that when this shrinking operation is applied in parallel to all pixels in an image, only 1s which do not disconnect a component will be erased and that 0s do not become 1s when this would connect previously unconnected components. The shrinking operation has the effect of squeezing each connected component into the lower righthand corner of its bounding box until only 1 pixel remains, which is then deleted by the next shrinking operation. An example is shown in the lower part of Figure 3. The number of shrinking operations required to shrink an object until it contains only 1 pixel is at most the distance from the lower righthand corner of the object's bounding box to the most distant pixel in the object, where the distance between the points is measured using the Manhattan metric: the distance from (x1,y1) to (x2,y2) is |x1-x2| + |y1-y2|. As a result, every connected component will have disappeared after 2N shrinking operations.

Levialdi uses the shrinking operation to count the number of connected components. In his algorithnm, whenever a connected component disappears, a special marker is created which then moves to the lower righthand corner of the array. Whenever two special markers arrive at the same location, a new marker which represents the sum of the previous markers is created. The marker which arrives at the lower righthand corner after 2N iterations represents the number of connected components in the image.

The component shrinking algorithm is based on Levialdi's shrinking operation and operates in two phases. In the first phase, Levialdi's shrinking operation is applied in parallel to the entire image 2N times. After each shrinking operation, a different image is obtained. The result of applying the shrinking operation y times to the original image will be called "partial result y". Assume that partial result y is stored in memory location y in the PEs.

In the second phase, the labels are assigned by examining the partial results in reverse order, starting with the empty image that resulted from the final shrinking operation. Stage y of the second phase, y ranging from 2N to 1, consists of first transferring the label from each PE(i,j) having a 1 in memory location y to those PEs (i,j), (i-1,j), (i,j+1) and (i-1,j+1) having a 1 in memory location y-1. Call this the "first assignment." Next, any PE (i,j) which has a 1 in memory location y-1 and which has not received a label generates a new label which is the concatenation of the numbers y, i and j. Call this the "second assignment." After processing all values of y from 2N to 1, each connected component will have a unique label. This can be seen by noting that a new label is created for exactly those pixels which became isolated 1s during the shrinking process. Because every component is shrunk to an isolated 1 which exists for only one stage, there is a unique label created by the second assignment, for every connected component. The label for a component is transferred from stage y to stage y-1 in a way that insures that it is sent to all pixels at stage y-1 which correspond to the same component at stage y, and to no others, as will now be shown.

It should be evident by inspection of the shrinking rules that every PE with a (nonisolated) 1 at stage y-1 receives one or more labels in the first assignment. If these are all the same then it is the label assigned to the same component at stage y and, by induction, the label is correct. What remains is showing that the labels received in the first assignment are all the same. If a PE were to get different labels, then it would be part of a single component in partial result y-1 that shrank to different isolated 1s. Because this is impossible given Levialdi's rules, the labels received by a processor during the first assignment must be consistent.

The worst case time requirement for the "component shrinking algorithm" is less than that required for the "component broadcasting algorithm" because the shrinking algorithm allows labels to pass through PEs that hold image pixels not belonging to the component. In contrast, the broadcasting algorithm sends labels only to PEs which hold pixels belonging to the component, so the labels must follow the contours of the components to which they belong. As the spiral in Figure 2 demonstrates, this is very slow in the worst case.

## 2.2 Log Space Connected Component Labeling on a Mesh

An important limitation of the component shrinking algorithm is that it requires O(N) bits of memory per PE, while the component broadcasting algorithm requires only O(logN) bits of memory per PE. However, the component shrinking algorithm can be modified so that it too requires only O(log n) bits of memory per PE. The resulting algorithm, which will be called the "log component shrinking algorithm", also requires O(N log N) time in the worst case.

The log component shrinking algorithm is the same as the component shrinking algorithm except that only log(N)+2 partial results from the shrinking operations are stored. The major difference between the algorithms is that in the original algorithm every partial result y was stored, but in this algorithm many of the partial results are not stored and so they must be calculated. Since the second phase partial results are processed in order from the last to the first, it would be convenient if partial result y-1 could be calculated from partial result y. Unfortunately, this is not the case. Instead, the log(N) + 2 stored partial results must be used judiciously. The technique used is to store a few of the partial results – those positioned at approximately 1/2, 3/4, 7/8, 15/16, etc. of the way through the sequence – and then to recreate the missing ones. The exact rules specifying how the results are stored are given below.

Adopt the following notation:

- $y_k$ is the k-th bit of the binary representation of the number y, where the least significant bit is the 0-th bit. For example, $y_1 = 0$ if y = 5.
- The binary representation of a nonnegative integer is written by listing the bits within parentheses separated by commas. For example, $y = (y_m, y_{m-1}, ...y_0)$ is an m + 1 bit representation of v.
- Last(y) is the bit position of the rightmost 1 in the binary representation of y, with bit position 0 being the least significant bit. Last(0) is undefined. For example, Last(12) = 2.
- Flip(y, j) is the number with the binary representation that is the binary representation of y with bit j complemented. For example, Flip(7,1) = 5.
- Result_In(j) = y when partial result y is stored in memory location j. Notice that the value of Result_In(j) will depend on when during the algorithm it is evaluated.

Figure 4 shows some values for the functions Last(y) and Flip(y, Last(y)).

Using these definitions, it is possible to define how the algorithm uses the O(log(N)) bits of memory per PE. When partial result y is calculated during the first phase, it is stored in memory location j where j = Last(y). Then in the second phase, after assigning the labels for partial result y, partial result x is required, where x = y - 1. Partial result x is calculated by retrieving partial result v from memory, where v ≤ x, and applying x - v shrinking operations to it. The results of these shrinking operations are stored in the appropriate memory locations (partial result w is stored in location Last (w)). In the log component shrinking algorithm, v = Flip(y, Last(y)). It will be shown that by retrieving partial result v where v is defined in this manner, only O(N log N) shrinking operations are required during the second phase. As a result, the algorithm operates in O(N log N) time. Figure 5 shows the contents of the variables v, w, x and y after each call to the Shrink function, assuming that N = 4. In addition, the partial result which is located in each of the 4 image memory locations is shown. Note that there are no entries for odd values of y because when y is odd, partial result y-1 is available without performing any additional shrinking operations. The fact that partial result v is present in memory when partial result x is needed is treated below. A pseudo-code representation of the algorithm and an analysis of its complexity follow.

The algorithm is specified using a modified C language syntax. The keyword "PE" in a variable declaration indicates that every PE has a copy of the variable, while variables declared without the "PE" keyword are stored in the controller. The "where...elsewhere" statement is a generalization of the "if...else" statement that allows some of the PEs to perform one set of operations and the remaining PEs to perform a different set of operations. The "Shift(variable, direction)" statement transfers the given variable 1 PE in the given direction. PEs on the edge of the mesh that would not receive data during a Shift are given a 0. It is assumed that N is a power of 2. There is no code for the first phase of the algorithm, because when y = 2N, v = 0 (because N is a power of 2), so partial results 1 through 2N-1 are calculated when y = 2N. This is exactly the first phase. The code for the main routine of the log component shrinking algorithm is given in Figure 6. The code for a number of supporting routines is given in Figure 7.

## 2.3 Algorithm Correctness

In the log component shrinking algorithm, after assigning the labels for partial result y, it is necessary to calculate partial result x, where x = y-1. This is done by retrieving partial result v, where v = Flip(y,Last(y)). If v = 0, this result is retrieved from memory location log(N) + 1, because of the initial assignment of the original image to that location. If v ≠ 0, this result is retrieved from memory location Last(v). The fact that Last(v) actually does hold partial result v, provided that v ≠ 0, rests on the following claim.

*CLAIM*: When the labels for partial result s are assigned in the log component shrinking algorithm, for all $k, 0 \leq k \leq l$, if $s_k = 1$ then $z$ = Result_In(k) = $(s_l, s_{l-1}, ...s_k, z_{k-1}, ..., z_0)$ where $z_i = 0$ for $0 \leq i \leq k-1$.

*PROOF*: Omitted.

## 2.4 Time Analysis

The asymptotic complexity of the algorithm is governed by the number of times the Shrink function in the inner for-loop is executed. Let this number be T(N), let n = 2N and let m = log(n). Then:

$$\begin{aligned} T(N) &= \textstyle\sum_{i=1}^{n}(2^{Last(i)} - 1) \\ &= (\textstyle\sum_{i=1}^{n} 2^{Last(i)}) - n \\ &= (\textstyle\sum_{i=0}^{m-1} 2^{-i-1} n 2^{i}) + n - n \\ &= \tfrac{1}{2}\textstyle\sum_{i=0}^{m-1} n \\ &= \tfrac{1}{2} nm \\ &= N(logN + 1) = O(NlogN) \end{aligned}$$

Line 1 follows from the fact that the variable w varies from v+1 through y-1, and the fact that $v = Flip(y, Last(y)) =$

$y - 2^{Last(y)}$, so for any value of y, the inner for-loop is executed $2^{Last(y)}$ - 1 times. Line 3 follows from line 2 by noting that there are $2^{-i-1}n$ numbers $x$ where $1 \leq x < n$ such that $Last(x) = i$ for each $i$, $0 \leq i \leq m$ -1 (recall that $n = 2^m$). The additional "+n" term corresponds to $x = n$, so $Last(x) = m$.

## 3 Summary

Two new local algorithms have been presented for the component labeling problem, the first requiring O(N log N) bit operations in worst case and O(N) bits of memory per PE and the second having the same time complexity, but requiring O(log N) bits of memory per PE. These bounds improve on known local algorithms by a factor of N. Although it seems likely that the techniques presented in [17] can be used to obtain an O(N) time algorithm, such an algorithm is expected to be slower than those given here for practical values of N. The two new algorithms are thus the first practical algorithms for connected component labelling.

## 4 Acknowledgement

The work of R. Cypher was supported by a National Science Foundation Fellowship and L. Snyder was supported in part by the National Science Foundation Grant DCR 8416878.

## References

[1] A. P. Reeves, "SURVEY: Parallel Computer Architectures for Image Processing", *Computer Vision, Graphics, and Image Processing*, 25:68-88, 1984.

[2] T. J. Fountain, "Array Architectures for Iconic and Symbolic Image Processing," *Proceedings 8th International Conference on Pattern Recognition*, pp. 24-33, 1986.

[3] M. J. B. Duff, ed., *Intermediate-Level Image Processing*, Academic Press, London, 1986.

[4] Hussein A. H. Ibrahim, J. R. Kender, D. Shaw, "The Analysis and Performance of Two Middle-Level Vision Tasks on a Fine-Grained SIMD Tree Machine", *Proceedings of the Conference on Computer Vision and Pattern Recognition*, IEEE, pp. 248-256, 1985.

[5] A. Rosenfeld, A. Kak, *Digital Picture Processing*, Academic Press, vols. 1-2, 1982.

[6] A. Agraval, A. Kulkarni, "A Sequential Approach to the Extraction of Shape Features", *CVGIP*, 6, pp. 538-557, 1977.

[7] F. Veillon, "One Pass Computation of Morphological and Geometrical Properties of Objects in Digital Pictures", *Signal Processing*, 1:175-189, 1979.

[8] R. Hummel, A. Rojer, "Implementing a Parallel Connected Component Algorithm on MIMD Architectures", Workshop on Computer Architecture for Pattern Analysis and Image Data Base Management, Miami, Florida, IEEE, 1985.

[9] D. Nassimi, S. Sahni, "Finding Connected Components and Connected Ones on a Mesh-Connected Parallel Computer", *SIAM Journal of Computing*, 9(4):744-757, 1980.

[10] D. Hillis, G. Steele, "Data Parallel Algorithms," *Communications of the ACM*, 29(12), December 1986.

[11] M. J. B. Duff, "Review of the CLIP Image Processing System", National Computer Conference, Anaheim, California, 1978.

[12] Kenneth E. Batcher, "Design of a Massively Parallel Processor," *Transactions on Computers*, IEEE C-29(9):836-840, 1980.

[13] J. L. Potter, "Image Processing on the Massively Parallel Processor", *Computer*, pp. 62-67, 1983.

[14] C. Dyer, A. Rosenfeld, "Parallel Image Processing by Memory-Augmented Cellular Automata", *Transactions on Pattern Analysis and Machine Intelligence*, Vol. PAMI-3(1):29-41, 1981.

[15] S. R. Kosaraju, "Fast parallel processing array algorithms for some graph problems", *Proceedings of the 11th Annual ACM Symposium on the Theory of Computing*, pp. 231-236, 1979.

[16] S. Levialdi, "On Shrinking Binary Picture Patterns", *Communications of the ACM*, 15(1):7-10, 1972.

[17] Quentin F. Stout, "Using Clerks in Parallel Processing", *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, 272-279, 1983.

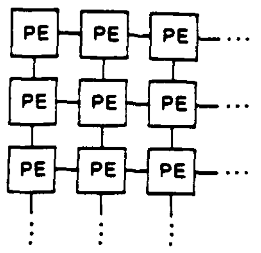

Figure 1. Mesh connected computer.

| Y | Last (Y) | Flip (Y Last (Y)) |
|---|---|---|
| 1 | 0 | 0 |
| 2 | 1 | 0 |
| 3 | 0 | 2 |
| 4 | 2 | 0 |
| 5 | 0 | 4 |
| 6 | 1 | 4 |
| 7 | 0 | 6 |
| 8 | 3 | 0 |

Figure 4. Table for LAST and FLIP operators.

```
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1
1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1
1 0 1 1 1 1 1 1 1 1 1 1 0 1 0 1
1 0 1 0 0 0 0 0 0 0 0 1 0 1 0 1
1 0 1 0 1 1 1 1 1 1 0 1 0 1 0 1
1 0 1 0 1 0 0 0 0 1 0 1 0 1 0 1
1 0 1 0 1 0 1 0 0 1 0 1 0 1 0 1
1 0 1 0 1 0 1 1 1 1 0 1 0 1 0 1
1 0 1 0 1 0 0 0 0 0 0 1 0 1 0 1
1 0 1 0 1 1 1 1 1 1 1 1 0 1 0 1
1 0 1 0 0 0 0 0 0 0 0 0 0 1 0 1
1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

Figure 2. Spiral connected component.

|  |  |  |  | Partial Result in Memory Location | | | |
|---|---|---|---|---|---|---|---|
| V | W | X | Y | 3 | 2 | 1 | 0 |
| 0 | 1 | 7 | 8 | 0 |  |  | 1 |
| 0 | 2 | 7 | 8 | 0 |  | 2 | 1 |
| 0 | 3 | 7 | 8 | 0 |  | 2 | 3 |
| 0 | 4 | 7 | 8 | 0 | 4 | 2 | 3 |
| 0 | 5 | 7 | 8 | 0 | 4 | 2 | 5 |
| 0 | 6 | 7 | 8 | 0 | 4 | 6 | 5 |
| 0 | 7 | 7 | 8 | 0 | 4 | 6 | 7 |
| 4 | 5 | 5 | 6 | 0 | 4 | 6 | 5 |
| 0 | 1 | 3 | 4 | 0 | 4 | 6 | 1 |
| 0 | 2 | 3 | 4 | 0 | 4 | 2 | 1 |
| 0 | 3 | 3 | 4 | 0 | 4 | 2 | 3 |
| 0 | 1 | 1 | 2 | 0 | 4 | 2 | 1 |

Figure 5. Memory contents after each call to the shrink function.

```
0 0    0 0    0 0    0 1    1 0    1 0    1 1
0 0    0 1    1 0    0 0    0 0    1 0    0 0


0 0 0 0 0    0 0 0 0 0    0 0 0 0 0    0 0 0 0 0
0 0 1 1 0    0 0 0 1 0    0 0 0 0 0    0 0 0 0 0
0 1 1 0 0    0 0 1 1 0    0 0 0 1 0    0 0 0 0 0
0 0 0 0 0    0 0 0 0 0    0 0 0 0 0    0 0 0 0 0
   (A)          (B)          (C)          (D)
```

Figure 3. Upper Part: configurations for Levialdi's shrinking algorithm. Lower Part: example of component shrinking.

**Algorithm**

```
#define NOLABEL 0
/* function Log_Space_Label returns the connected component labels */
/* for the given original image: */

PE int Log_Space_Label(original_image)
PE int original_image;          /* image to be labeled */
{
int v, w, x, y;               /* controller variables holding */
                              /* result numbers */
PE int image[log(N)+2];       /* O(log(N)) memory for holding */
                              /* partial results */
PE int old_label, new_label   /*previous and current labels */

     image[log(N)+1] = original_image;
     old_label = NOLABEL;
     for (y = 2*N; y >= 1; --y)
     {
          x = y-1;
          v = Flip(y,Last(y));
          for (w = v+1; w <= x; ++w)
          {
               if (w-1 == 0)
               {
                    image[Last(w)] = Shrink(image[log(N)+1]);
               }
               else
               {
                    image[Last(w)] = Shrink(image[Last(w-1)]);
               }
          }
     if (x == 0)
          new_label = Label(old_label,image[log(N)+1],y);
     else
          new_label = Label(old_label,image[Last(x)],y);
     old_label = new_label;
     }
     return(new_label);
}
```

Figure 6. Log component shrinking algorithm.

## Algorithm Subroutines

```
#define NOLABEL 0


PE int Heaviside(x)
PE int x;
{
PE int answer;


     where (x > 0)
          answer = 1;
     elsewhere
          answer = 0;
     return(answer);
}



PE int Shrink(image)
PE int image;
{
PE int here, north, west, northwest;


     here = image;
     Shift(image,DOWN);
     north = image;
     image = here;
     Shift(image,RIGHT);
     west = image;
     Shift(image,DOWN);
     northwest = image;
     return(Heaviside(Heaviside(here+west
            +north-1)+Heaviside(here+
            northwest-1)));
}
```

```
/*function Label returns the labels
     for partial result y-1: */

PE int Label(old_label,new_image,y)
PE int old_label;  /*labels for partial
                          result y */
PE int new_image;  /*partial result y-1 */
PE int y;
{
PE int answer, here, south, east, southeast;


   here = old_label;
   Shift(old_label,UP);
   south = old_label;
   old_label = here;
   Shift(old_label,LEFT);
   east = old_label;
   Shift(old_label,UP);
   southeast = old_label;
   where (new_image == 1)
   {
      where (here != NOLABEL)
         answer = here;
      elsewhere where (south != NOLABEL)
         answer = south;
      elsewhere where (east != NOLABEL)
         answer = east;
      elsewhere where (southeast != NOLABEL)
         answer = southeast;
      elsewhere
         answer = y*N**2+pe_row*N+pe_col;
   }
   elsewhere
   {
      answer = NOLABEL;
   }
   return(answer);
}
```

Figure 7. Algorithm Subroutines.