DTIC FILE COPY

(4)

AD-A197 574

Unclassified

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) Hypercube and Shuffle-Exchange Algorithms for Image Component Labeling	5. TYPE OF REPORT & PERIOD COVERED Technical Report
	6. PERFORMING ORG. REPORT NUMBER TR 87-04-03
7. AUTHOR(s) R. E. Cypher, J. L. C. Sanz 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 April 1987
	13. NUMBER OF PAGES 6
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)

DTIC ELECTE JUL 25 1988 S D H

18. SUPPLEMENTARY NOTES

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

parallel algorithms, image processing, hypercube, shuffle-exchange, connected component labeling

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

This paper presents algorithms for labeling the connected components of a binary image using a hypercube or shuffle-exchange computer. The algorithms label the components of an $N^{1/2} \times N^{1/2}$ pixel image in $O(\log^2 N)$ time using a hypercube or shuffle-exchange computer with N processors and a constant amount of memory per processor. The algorithms that are presented are the first to solve this problem in $O(\log^2 N)$ time. The algorithms are based on a divide-and-conquer approach and use as a subroutine an $O(\log N)$ time PRAM algorithm for labeling the connected components of a graph. The simulation of the PRAM

DD FORM 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

by the hypercube and shuffle-exchange computers is particularly efficient because the PRAM that is being simulated has only $O(N^{3/4})$ processors and memory cells.

DTIC
COPY
INSPECTED
6

Accession For
NTIS GRA&I ☑
DTIC TAB ☐
Unannounced ☐
Justification
By
Distribution/
Availability Codes
Dist | Avail and/or special
A-1

Hypercube and Shuffle-Exchange Algorithms for Image Component Labeling

R. Cypher (*) J.L.C. Sanz (**) L. Snyder (*)

(*) Computer Science Department
University of Washington - Seattle, Wa.

(**) Computer Science Department
IBM Almaden Research Center - San Jose, Ca.

Abstract

This paper presents algorithms for labeling the connected components of a binary image using a hypercube or shuffle-exchange computer. The algorithms label the components of an $N^{1/2} \times N^{1/2}$ pixel image in $O(\log^2 N)$ time using a hypercube or shuffle-exchange computer with N processors and a constant amount of memory per processor. The algorithms that are presented are the first to solve this problem in $O(\log^2 N)$ time. The algorithms are based on a divide-and-conquer approach and use as a subroutine an $O(\log N)$ time PRAM algorithm for labeling the connected components of a graph. The simulation of the PRAM by the hypercube and shuffle-exchange computers is particularly efficient because the PRAM that is being simulated has only $O(N^{3/4})$ processors and memory cells.

1.- Introduction

The problem of labeling the connected components in a binary image is addressed. This problem is of great importance to the image processing community because it forms a bridge between low-level iconic algorithms and high-level symbolic ones [13],[17]. Because of its importance, a large number of parallel algorithms have been developed for image component labeling [3],[4],[5],[6],[7],[8],[9],[12],[16]. This paper presents algorithms for image component labeling using hypercube and shuffle-exchange computers. These algorithms are asymptotically faster than previously known algorithms for the problem using these types of parallel computers. The remainder of this section examines models of parallel computers, the image component labeling problem and previous work in the area. Section 2 describes some previously published algorithms that will be used as subroutines. In Section 3, the hypercube and shuffle-exchange algorithms are presented and analysed. Section 4 contains conclusions.

The hypercube, the shuffle-exchange and the PRAM are all models of parallel computers. All of these models operate in a synchronous, SIMD manner. Let the processors in each model be numbered 0 .. N-1. In the hypercube and shuffle-exchange computers each processor has a constant amount of local random-access memory and can communicate with other processors through a fixed interconnection network. In the hypercube, processor i is connected to processor j if the binary representations of i and j differ in exactly 1 bit position. In the shuffle-exchange, processor i is connected to processor j if j = Shuffle(i), j = Unshuffle(i) or j = Exchange(i) where Shuffle(i) = 2i mod (N-1), Unshuffle is the inverse of Shuffle, and Exchange(i) = i+1 - 2(i mod 2) [15]. A hypercube or shuffle-exchange of size N is a hypercube or shuffle-exchange computer with N processors.

In the PRAM, all processors have access to a common memory. In a single time step, all processors can read from or write to the common memory. A PRAM of size N is a PRAM computer with N processors and N words of memory. Variants of the PRAM model differ in allowing multiple processors simultaneous access to a single memory location. The most powerful type of PRAM is the CRCW (concurrent read, concurrent write) PRAM. In this model, any number of processors may simultaneously read from or write to a single memory location. When more than one processor tries to read from a single memory location, all of them succeed. When more than one processor tries to write to a single memory location, one of them succeeds. The selection of which processor succeeds depends on which type of CRCW PRAM model is being used.

The input to the image component labeling problem is an $N^{1/2} \times N^{1/2}$ array of binary pixels. Two 1-valued pixels are *adjacent* if they share a vertical or horizontal edge, and they are *connected* if there exists a path of adjacent 1-valued pixels from one to the other. The image component labeling problem

is to label each 1-valued pixel such that any two 1-valued pixels receive the same label if and only if they are connected. A set of pixels that must receive the same label is a *connected component* of the image.

The image component labeling problem is a special case of the graph component labeling problem. Specifically, given a binary image I, create the corresponding undirected graph G = (V, E) where V consists of the 1-valued pixels in I and E consists of all pairs (i, j) where i and j are adjacent 1-valued pixels in I. The connected components in G correspond exactly to the connected components in I. As a result of this correspondence, graph component labeling algorithms can be used to solve the image component labeling problem.

In [14], Shiloach and Vishkin present a PRAM algorithm for labeling the connected components of an undirected graph containing v vertices and e edges. Their algorithm requires O(log v) time on a CRCW PRAM of size v + 2e. The type of CRCW PRAM that they use will be called the Arbitrary-CRCW PRAM. In this model, the processor that succeeds in writing to a contested memory location is chosen arbitrarily. By using the correspondence between graph and image component labeling that was discussed above, it is clear that Shiloach and Vishkin's algorithm can be used to obtain an algorithm for labeling the connected components of an $N^{1/2} \times N^{1/2}$ image in O(log N) time using an Arbitrary-CRCW PRAM of size N.

In [10], Nassimi and Sahni present an algorithm for simulating a PRAM with a hypercube or shuffle-exchange computer. Their algorithm simulates a single operation of an Arbitrary-CRCW PRAM of size N in $O(\log^2 N)$ time using a hypercube or shuffle-exchange of size N. Using this simulation and the PRAM algorithm mentioned above, it is possible to obtain an $O(\log^3 N)$ time algorithm for labeling the connected components of an image with a hypercube or shuffle-exchange computer. The algorithms presented in this paper improve upon this result by requiring only $O(\log^2 N)$ time. They are the first $O(\log^2 N)$ time algorithms for labeling the connected components of an image with a hypercube or shuffle-exchange computer.

2.- *Subroutines*

There are a number of previously published algorithms that are used as subroutines in the current paper. This section briefly discusses these subroutines. One useful subroutine consists of the Rank and Concentrate algorithms presented in [10]. This subroutine is used to consolidate data in a hypercube or shuffle-exchange computer. The input is a set of R records, stored no more than one per processor, in an N processor machine. The output is the same set of records, now stored one per processor, in the first R processors. Nassimi and Sahni present an O(log N) time algorithm for this operation. A slight generalization of this problem starts with R records, stored no more than K per processor, and returns with the records stored in the first ceiling(R/K) processors, again having no more than K records per processor. For any fixed value of K, this generalized version of the problem is easily solved in O(log N) time by simulating a KN processor machine with an N processor machine and running Nassimi and Sahni's algorithm on the simulated machine. This generalized algorithm will be called the Compress(K) algorithm.

The paper that presents the Rank and Concentrate algorithms shows how a single operation of an Arbitrary-CRCW PRAM of size N can be simulated in $O(\log^2 N)$ time by a hypercube or shuffle-exchange of size N. The simulation algorithm consists of an $O(\log^2 N)$ time bitonic sort and a number of O(log N) time routines. Thus the $O(\log^2 N)$ time of the simulation algorithm is due solely to the time required for the bitonic sort.

Another important subroutine is also given by Nassimi and Sahni. In [11], they present algorithms for sorting data on hypercube and shuffle-exchange computers. For any fixed value of $K \geq 1$, the least-significant-digit radix sort algorithm that they present sorts N integers, each of which is in the range $1 .. N^{1+1/K}$, in O(log N) time using a hypercube or shuffle-exchange of size $N^{1+1/K}$. This algorithm is important because it can be used to simulate a single operation of an Arbitrary-CRCW PRAM of size N with a hypercube or shuffle-exchange of size $N^{1+1/K}$ in O(log N) time. This simulation is identical to the $O(\log^2 N)$ time simulation presented in [10], except that the bitonic sort is replaced by the least-significant-digit radix sort. This O(log N) time simulation algorithm will be referred to as the Fast_Simulation(K) algorithm.

The other subroutine that will be used in this paper is Shiloach and Vishkin's PRAM algorithm for labeling the connected components of a graph [14]. As was mentioned earlier, their algorithm uses an Arbitrary-CRCW PRAM of size v + 2e to label the connected components of a graph containing v vertices and e edges in O(log v) time. In order to describe their algorithm, a few terms must be defined. A *rooted tree* is a directed graph where: 1) the underlying undirected graph is a tree, and 2) there is a directed path from each vertex in the rooted tree to the root vertex. A *rooted star* is a rooted tree

Copy available to DTIC does not permit fully legible reproduction

in which every vertex (including the root) points directly to the root.

In the algorithm, each vertex i has associated with it a variable D(i) that points to some vertex in the graph. If the pairs (i, D(i)) are viewed as being directed edges, then the variables D(i) define a graph that is called the *pointer graph*. Throughout the algorithm, the pointer graph consists of a forest of rooted trees. At the start of the algorithm, D(i) = i for all i, so the pointer graph consists of v rooted stars each containing 1 vertex. The algorithm proceeds by performing a number of "shortcut" operations that reduce the heights of the rooted trees and "hooking" operations that merge rooted trees. At the end of the algorithm, the pointer graph consists of a forest of rooted stars, where each star contains the vertices of one of the connected components in the original graph. Thus the D(i) pointer fields can be considered to be labels that partition the graph into connected components. Because each shortcut and hooking operation requires constant time, and because Shiloach and Vishkin prove that only O(log v) shortcut and hooking operations are required, the entire algorithm runs in O(log v) time.

3.- *The Image Component Labeling Algorithms*

Having discussed the necessary subroutines, it is now possible to describe the $O(\log^2 N)$ time hypercube and shuffle-exchange algorithms for image component labeling. This section gives a brief description of the algorithms. This description is accompanied by an example that is presented in Figures 1-4. Then a more detailed description and a proof of correctness are given. Finally, an analysis of the running time is presented.

The hypercube and shuffle-exchange algorithms are very closely related to one another. Both use N processors to label the connected components in an $N^{1/2} \times N^{1/2}$ pixel binary image (see Figure 1). They are based on a divide-and-conquer technique where the N pixel image is divided into approximately $N^{1/2}$ square windows each containing approximately $N^{1/2}$ pixels. The connected components within the windows are labeled using a recursive call (see Figure 2).

After the windows have been labeled, adjacent 1-valued pixels have the same label unless they lie on the borders of different windows. The next task is to correct the labels of the pixels that lie on the borders of the windows. This is accomplished by using Shiloach and Vishkin's O(log N) time PRAM algorithm for labeling the connected components in a graph. This gives the border pixels their correct labels (see Figure 3). Then the non-border pixels are relabeled according to the labels that were assigned to the border pixels. This is the desired labeling of the components in the image (see Figure 4). Then the last step changes the labels of some of the components. This step is required in order to make the recursive call function correctly.

The details of the image component labeling algorithm are given below. It is assumed that $N = 4^n$. Each processor i has a variable D(i) that holds the current label of its pixel.

Step 1:

- If N = 1, set D(i) := i. Otherwise, recursively label the M x M windows of the image in parallel, where $M = 2^m$ and m = ceiling(n/2). This step sets the variable D(i) in all of the processors.

Step 2:

- Let S = the set of processors that are on the borders of the M x M windows and contain 1-valued pixels. For each i in S, processor i creates up to 4 edge records as follows. For each j in S, if i is adjacent to j and if i and j are in different M x M windows, then processor i creates the record <i, j>. If D(i) ≠ i, then processor i creates the records <i, D(i)> and <D(i), i>.

Step 3:

- The edge records are placed in the first N/M processors, with each processor holding at most 12 edge records. The Compress(12) subroutine is used to accomplish this.

Step 4:

- Shiloach and Vishkin's O(log N) time PRAM algorithm [14] is used to label the connected components of the graph represented by the edge records created in Step 2. The edge records form the input vector E and the processors in S correspond to the PRAM processors used in the algorithm. Notice that there are at most $12N^{3/4}$ edge records and $N^{3/4}$ processors in S. The Fast_Simulate(3) algorithm is used to simulate an Arbitrary-CRCW PRAM of size $O(N^{3/4})$ on the hypercube or shuffle-exchange of size N. This step relabels the border pixels.

Step 5:

- All processors (including those not in S) set D(i) := D(D(i)). At this point each connected component in the $N^{1/2} \times N^{1/2}$ image is represented by a rooted star. This step relabels the non-border pixels according to the labels that were assigned to the border pixels in Step 4.

Step 6:

- Let T = the set of processors that are on the border of the $N^{1/2} \times N^{1/2}$ image. For each i in T, processor i attempts to set D(D(i)) := i. For each variable D(D(i)) being written to, it is assumed that one (arbitrarily selected) write attempt succeeds. Then all processors set D(i) := D(D(i)). This step assures that the recursive call will work correctly.

After Step 1, any pair of pixels that have the same label are in fact in the same connected component. However, it is possible that more than one label has been assigned to a single connected component (this happens whenever a component appears in more than one window). The purpose of Steps 2-5 is to relabel pixels so that only 1 label is assigned to each connected component. It will be shown that Steps 2-5 do in fact accomplish this.

Let Label(i, k) be the label assigned to pixel i after Step k. Let G be the undirected graph created in Steps 2 and 3. Note that G = (V, E) where V = S and E is the set of all unordered pairs (i, j) where i and j are in S, and either j = Label(i, 1) or else i and j are adjacent and located in different M x M windows.

Claim 1: If Step 1 correctly labels the M x M windows, then for all i,j in S, Label(i, 4) = Label(j, 4) if and only if i and j are in the same connected component in the image.

Proof: (Omitted)

Claim 2: If Step 1 correctly labels the M x M windows, then for all 1-valued pixels i and j, Label(i, 5) = Label(j, 5) if and only if i and j are in the same connected component in the image.

Proof: (Omitted)

Claim 3: For all 1-valued pixels i and j, Label(i, 5) = Label(j, 5) if and only if i and j are in the same connected component in the image.

Proof: (Omitted)

Claim 3 shows that the image is correctly labeled after Step 5. Because Step 6 only relabels some of the components, and no two of the components are given the same new label, the image is also correctly labeled after Step 6. Step 6 insures that all of the image components that contain an element of T are rooted in T. This is necessary so that when the algorithm is called recursively in Step 1, the labels that are assigned guarantee that in Step 2, if i is in S then D(i) is also in S.

Step 2 requires constant time. Step 3 requires O(log N) time. Step 4 consists of O(log N) PRAM steps, each of which is simulated in O(log N) time, so Step 4 requires $O(\log^2 N)$ time. Steps 5 and 6 can each be considered to be a single operation of an Arbitrary-CRCW PRAM of size N, and can be simulated in $O(\log^2 N)$ time [10]. The total time for steps 2 through 6 is thus less than or equal to $C\log^2 N$ for some constant C, and the total time for the algorithm is thus $O(\log^2 N)$.

4.- Conclusion

This paper has shown that $O(\log^2 N)$ time image component labeling algorithms are possible on the hypercube and shuffle-exchange computers. The algorithms that are presented make use of a divide-and-conquer strategy. The image is divided into windows that are labeled recursively, and the results of these labelings are then combined to obtain the final result. The combination step uses a PRAM algorithm for labeling the connected components of a graph. The key to the algorithms is the reduction of the amount of data that must be routed during this combination step. This reduction is due to the fact that only the borders of the windows need to be considered.

These algorithms demonstrate that PRAM algorithms can be very helpful in designing fast algorithms for realistic parallel machines, but that they must be used carefully. A straightforward approach would simply simulate the PRAM algorithm on the hypercube or shuffle-exchange computer. This approach yields $O(\log^3 N)$ time algorithms. By using a divide-and-conquer approach and only simulating the PRAM algorithm when the amount of data to be processed is reduced, $O(\log^2 N)$ time algorithms are obtained.

Bibliography

1. R. Cypher, J. L. C. Sanz, L. Snyder, "EREW PRAM and Mesh Connected Computer Algorithms for Image Component Labeling", to appear in 1987 IEEE Computer Society Workshop on Computer Architecture, Pattern Analysis and Machine Intelligence.

2. R. Cypher, J. L. C. Sanz, L. Snyder, "Algorithms for Image Component Labeling on SIMD Mesh Connected Computers", to appear in Proc. 1987 Intl. Conference on Parallel Processing.

3. R. Hummel, "Connected Component Labelling in Image Processing with MIMD Architectures" in Intermediate-Level Image Processing, Academic Press, 1986, pp. 101-127.

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

5. Y. Hung, A. Rosenfeld, "Parallel Processing of Linear Quadtrees on a Mesh-Connected Computer", Tech. Rep. CAR-TR-278, Center for Automation Research, U. of Maryland, March 1987.

6. V. K. P. Kumar, M. M. Eshaghian, "Parallel Geometric Algorithms for Digitized Pictures on Mesh of Trees" (preliminary version), Proc. 1986 Intl. Conference on Parallel Processing, pp. 270-273.

7. W. Lim, "Fast algorithms for labeling connected components in 2-D arrays", Tech. Rep. 86.22, Thinking Machines Corp., Cambridge, Mass., July 1986.

8. R. Miller, Q. Stout, "Varying Diameter and Problem Size in Mesh-Connected Computers" (preliminary version), Proc. 1985 Intl. Conference on Parallel Processing, pp. 697-699.

9. D. Nassimi, S. Sahni, "Finding Connected Components and Connected Ones on a Mesh-Connected Parallel Computer", Siam J. Comput., vol. 9, no.4, November 1980, pp.744-757.

10. D. Nassimi, S. Sahni, "Data Broadcasting in SIMD Computers", IEEE Transactions on Computers, vol. c-30, no. 2, February 1981, pp. 101-107.

11. D. Nassimi, S. Sahni, "Parallel Permutation and Sorting Algorithms and a New Generalized Connection Network", Journal of the ACM, Vol. 29, No.3, July 1982, pp. 642-667.

12. A. Rosenfeld, "Parallel Image Processing Using Cellular Arrays", IEEE Computer, pp. 14-20, 1983.

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

14. Y. Shiloach, U. Vishkin, "An O(log n) Parallel Connectivity Algorithm", Journal of Algorithms, vol. 3, 1982, pp. 57-67.

15. H. S. Stone, "Parallel Processing with the Perfect Shuffle", IEEE Transaction on Computers, vol. c-20, no. 2, February 1971, pp. 153-161.

16. Q. F. Stout, "Properties of Divide-and-Conquer Algorithms for Image Processing", 1985 IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Image Database Management, pp. 203-209, 1985.

17. S. Tanimoto, "Architectural Issues for Intermediate-Level Vision" in Intermediate-Level Image Processing, Academic Press, 1986, pp. 3-16.

Acknowledgement

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

0	1	1	0	0	0	1	1
0	1	1	1	1	0	1	1
1	0	1	0	1	1	1	0
1	0	1	0	0	0	0	1
1	0	1	0	1	1	0	1
0	0	1	0	0	0	0	1
1	0	1	1	1	1	1	1
1	1	0	0	0	0	1	1

Figure 1

0	2	2	2	[illegible]	[illegible]	[illegible]	[illegible]
0	2	2	2	[illegible]	[illegible]	[illegible]	[illegible]
17	0	2	0	[illegible]	[illegible]	[illegible]	[illegible]
17	0	2	0	0	[illegible]	[illegible]	[illegible]
33	0	35	0	37	37	[illegible]	[illegible]
0	0	35	0	0	[illegible]	[illegible]	[illegible]
49	0	35	35	[illegible]	[illegible]	[illegible]	[illegible]
49	49	0	0	0	[illegible]	[illegible]	[illegible]

Figure 2

0	2	2	0	0	0	2	2
0	2	2	2	2	0	7	2
17	0	2	0	2	7	7	0
17	0	2	0	0	0	0	2
17	0	2	0	37	37	0	2
0	0	35	0	0	0	0	2
49	0	35	2	2	40	40	2
49	49	0	0	0	0	2	2

Figure 3

0	2	2	0	0	0	2	2
0	2	2	2	2	0	2	2
17	0	2	0	2	2	2	0
17	0	2	0	0	0	0	2
17	0	2	0	37	37	0	2
0	0	2	0	0	0	0	2
49	0	2	2	2	2	2	2
49	49	0	0	0	0	2	2

Figure 4

END
DATE
9-88
DTIC