Jing-Fu Jenq
University of Minnesota
and
Sartaj Sahni
University of Florida

Abstract

We develop an $O(n^2)$ time serial algorithm to obtain the medial axis transform (MAT) of an $n \times n$ image. An $O(\log n)$ PRAM and an $O(\log^2 n)$ hypercube parallel algorithm for the MAT are also developed. Both of these use $O(n^2)$ processors. Two problems associated with the MAT are also studied. These are the area and perimeter reporting problem. We develop an $O(\log n)$ time hypercube algorithm for both of these problems. Here n is the number of squares in the MAT and the algorithms use $O(n^2)$ processors.

Keywords and Phrases

medial axis transform, image processing, serial and parallel algorithms, hypercube algorithms

* This research was supported in part by the National Science Fundation under grants DCR-84-20935 and MIP 86-17374

1 Introduction

The medial axis transform (MAT) is an image representation scheme proposed by Blum [BLUM67]. For simplicity, assume that we have an $n \times n$ digital image I with $I[i,j] \in \{0,1\}$, $0 \le i < n$, $0 \le j < n$. The essential idea in the MAT is to find a minimal set of upright squares whose union corresponds exactly to the regions in I that have value 1 (see Figure 1). While in [ROSE82] the squares are restricted to be of odd length, this restriction is really unnecessary and actually the purposes of the application are just as well served by using any set of regular shapes (eg. rectangles) that can be compactly represented and easily manipulated. Thus in this paper, as in [VO82], [CHAN87], and elsewhere, we place no restriction on the size of the squares. In passing, we note the problem of finding the minimum number of rectangles that cover the 1's in a binary image is NP-hard [GARE79].

In [VO82], an $O(kn^2)$ serial algorithm to obtain the MAT of an $n \times n$ binary image is presented. Here k is the size of the largest square in the MAT. In the worst case, k could be $O(n)$. So this algorithm has a worst case complexity that is $O(n^3)$. In Section 2, we present an $O(n^2)$ algorithm for this problem. It should be noted that the MAT of a planar shape with n sides can be obtained in $O(n\log n)$ time [LEE82]. Chandran and Mount [CHAN87] have developed an $O(n\log n)$ PRAM algorithm for the MAT. This algorithm uses $O(n^2)$ processors. In Section 3, we develop two PRAM algorithms for the MAT. Both use $O(n^2)$ processors and both have a run time of $O(\log n)$. The first uses $O(n^2\log n)$ space while the second uses $O(n^2)$ space. In Section 4 we map our second PRAM algorithm onto an n^2 processor hypercube. The resulting algorithm uses $O(1)$ memory per processor and has a time complexity that is $O(\log^2 n)$.

Area, perimeter, and contour of an image are some of the geometric properties that are commonly computed from its MAT representation. Since the algorithms for these properties work just as well on collections of rectangles (rather than just on collections of squares), these algorithms are generally developed for rectangle sets. $O(n\log n)$ serial algorithms to obtain the area and perimeter of a set of n rectangles are obtained in [WU86]. The contour of a set of n rectangles may be found in $O(nlogn + k)$ time where k is the number of segments in the contour [GUTI84] and [WOOD84]. Note that the contour may have $O(n^2)$ segments. Wu, Bhaskar, and Rosenfeld [WU88] have developed $O(n)$ time algorithms for both CREW PRAM and mesh connected computers to compute the contour, perimeter, and area of a set of n upright rectangles. Their PRAM algorithms uses $O(n)$ processors while their mesh connected computer algorithms use $O(n^2)$ processors. Chandran and Mount [CHAN87] obtain $O(\log n \log\log n)$ time CREW PRAM algorithms for the area and perimeter of n rectangles. They also obtain an $O(n)$ CREW PRAM a;gorithm for the contour problem. All their algorithms use $O(n)$ processors. Lu and Varman [LU88] develop an $O(\sqrt{n})$ parallel algorithm that computes the area of n rectangles on an n processor mesh connected computer. This algorithm is easily modified to run on an n processor hypercube in $O(\log^2 n)$ time. The algorithm may be further modified to compute the perimeter in $O(\log^2 n)$ time on an n processor hypercube. Using an n^2 processor CREW PRAM the contour can also be obtained in $O(\log n)$ time [JARV90]. The algorithm of [JARV90] can be modified to run on an n^2 processor hypercube in $O(\log^2 n)$ time. In Section 5 we present an $O(\log n)$ time algorithm to obtain the area on an n^2 processor hypercube. This algorithm is modified in Section 6 to obtain the perimeter in $O(\log n)$ time.

1	1	1					
	1	1	1	1			
		1	1	1	1	1	
			1	1	1	1	1
	1	1	1	1	1	1	1
	1	1	1	1		1	1
		1	1	1			
		1	1	1	1		

Figure 1 An 8×8 image

2 Serial Algorithm For MAT

Let $M[i,j]$ be the height of the largest square with top left corner $[i,j]$ all of whose image values are 1. In [VO82] and [CHAN87] it was observed that

$$M[i,j] = \begin{cases} 0 & \text{if } I[i,j] \neq 1 \\ \min\{M[i,j+1],\ M[i+1,j],\ M[i+1,j+1]\} + 1 & \text{if } I[i,j] = 1 \end{cases} \qquad (1)$$

In Eq(1), we assume that $M[n,j] = M[j,n] = 0,\ 0 \le j < n$. In [VO82] and [CHAN87], (1) is solved using an iterative approach that is based on a linked list of active pixels with value 1. (1) may, however, be solved in $O(n^2)$ time by computing $M[i,j]$ in order of the antidiagonals of M (Figure 2). The corresponding algorithm is given in Figure 3.

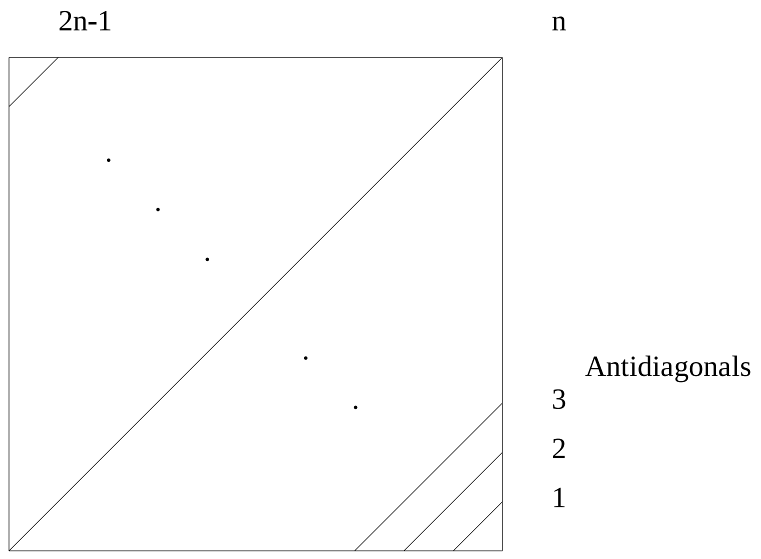

Figure 2 Antidiagonals of M numbered in computation order

The top left corners of the squares in the MAT can now be identified using equation (2) below . This assumes that $M[-1,j] = M[j,-1] = 0,\ 0 \le j < n$

$$T[i,j] = \begin{cases} \mathit{true} & \text{if } \max\{M[i,j-1], M[i-1,j], M[i-1,j-1]\} \le M[i,j] \\ \mathit{false} & \text{otherwise} \end{cases} \qquad (2)$$

Figure 4 gives the M values computed by the algorithm of Figure 3 using the image of Figure 1 as data. The top left corners of the squares in the MAT are identified by an asterisk (*). The size of the squares is given by the $M[i,j]$ value of the top left corner.

3 PRAM Algoirthms For MAT

Our $O(\log n)$ time n^2 processor CREW PRAM algorithms for the MAT of an $n \times n$ binary image I work in three steps as below.

Step 1 Compute $R[i,j]$ = max value of l such that

$I[i,j+k] = 1,\ 0 \le k < l,\ R[i,j] = 0$ if $I[i,j] = 0$

I.e., $A[i,j]$ is the number of consecutive 1's beginning at

$I[i,j]$ and extending rightward in row i

Step 2 Compute $M[i,j]$

Step 3 Compute $T[i,j]$

Since T is easily computed from M in $O(1)$ time using n^2 processors, we consider only the computation of R and M.

3.1 Computation of *R*

The steps in the computation of R are given in Figure 5. Figure 6 gives the values of Z, P, $LAST$, and R for the image of Figure 1. Steps 1 and 3 can be done in $O(1)$ time using one processor per $[i,j]$. Step 2 can be done in $O(logn)$ time using n processors per row of I [KRUS85]. Step 4 can be done in $O(logn)$ time using binary search and one processor per $[i,j]$. This step requires the concurrent read capability of the CREW PRAM. The overall complexity of the algorithm of compute R is $O(logn)$ and n^2 processors are used. The space requirement is $O(n^2)$.

3.2 Computation of M

3.2.1 $O(n^2 \log n)$ space method

In this method we first construct a table $MaxWidth[i,j,k]$, $0 \le i,j < n$, $0 \le k \le \log_2 n$ such that $MaxWidth[i,j,k]$ is the maximum width of a rectangle whose top left corner is $[i,j]$, height is 2^k, and all image points in the rectangle (including boundaries) are 1. This can be done in $O(logn)$ time by assigning one processor to each $[i,j]$ and having the assigned processor perform Step 1 of Figure 7. For simplicity, this assumes that $MaxWidth[i,*,*] = 0$ for $i \ge n$.

Step 2 of Figure 7 computes M from $MaxWidth$. One processor is assigned to compute each $M[i,j]$. We obtain $M[i,j]$ by first determining the maxmium length rectangle with top left corner at $[i,j]$ and such that its width is $\ge$ its height. $M[i,j]$ is the height of this rectangle. We start with $ht = 0$ and $width = \infty$ and then increase ht in steps that are a power of 2 while maintaining the invariant $width \ge ht$. Thus, if $M[i,j] = 13$, ht will go through the values 8, 4, and 1. The correctness of Step 2 is easily established. Since r is a strictly decreasing variable, the number of iterations of the **while** loop is $O(logn)$. Hence with n^2 processors a CREW PRAM can perform the algorithm of Figure 7 in $O(logn)$ time.

3.2.2 $O(n^2)$ space method

This time we compute only some of the entries in $MaxWidth$. Define the matrix MW as below.

$MW[q,j,k] =$ maximum width of a rectangle of ones with height 2^k and top left corner at $[q * 2^k, j]$, $0 \le q < n/2^k$, $0 \le k \le logn$

Figure 8 gives the values of q and k for which $MW[q,j,k]$ is computed for every column j of a 16×16 image. Each box represents one such $MW[q,j,k]$ value. The value of q is given in the box and the value of k is given at the top of each column of boxes. One may verify that $MW[q,j,k] = MaxWidth[q * 2^k, j, k]$. For any fixed j, the number of entries MW is $n(k = 1) + n/2\ (k = 2) + n/4 + ... + 1(k = \log_2 n) = O(n)$. So, the total number of entries in MW is $O(n^2)$.

Step 1 of Figure 9 computes MW. This step is easily executed in $O(logn)$ time by an n^2 processor PRAM. $M[i,j]$ is computed in step 2 by making two passes over $MW[*,j,*]$. In both passes we maintain the invariant $width \ge ht$. In the forward pass the table of Figure 8 is

```
for $k := 1$ to $n$ do
{ antidiagonal $k$ }
 $i := n - 1;$
 $j := n - k;$
 { bottom position on antidiagonal }
 while $(j < n)$ do
 begin
   if $I[i,j] = 0$ then $M[i,j] = 0$
   else $M[i,j] = \min\{ M[i,j+1], M[i+1,j], M[i+1,j+1] \} + 1$
   $i := i - 1;$
   $j := j + 1;$ {next position on antidiagonal }
 end ;
for $k := n+1$ to $2n-1$ do
  {antidiagonal k }
 $j := 0;$
 $i := 2n-1-k;$
 { bottom position on antidiagonal }
 while $( i \geq 0 )$ do
 begin
   if $I[i,j] = 0$ then $M[i,j] = 0$
   else $M[i,j] = \min\{ M[i,j+1], M[i+1,j], M[i+1,j+1] \} + 1$
   $i := i - 1;$
   $j := j + 1;$ {next position on antidiagonal }
 end;
```

Figure 3 Algorithm to compute M

scanned right to left. $[a,q]$ indexes a square of Figure 8. Inside the forward pass **while** loop it is

1*	2*	1					
	1	2*	2*	1			
		1	3*	3*	2	1	
			2	2	2	2*	1
	2*	3*	2	1	1	2*	1
	1	3*	2	1		1	1
		2	2	1			
		1	1	1	1*		

Figure 4 Example of marking maximal squares

the case that the rectangle with top left corner $[i,j]$, height $ht + 2^q$ and width $\min\{width, MW\}$, has all ones. If a is odd, we increase the height and look for further increase from the square $[\lceil a/2 \rceil, q+1]$. If a is even, we see if the height can be increased by 2^{q+1}. For this we examine the square $[\lceil a/2 \rceil, q+1]$. The forward pass terminates when one of the following occurs:

(a) $q > \log_2 n$

(b) $a \geq \lfloor n/2^q \rfloor$

(c) $MW[a,j,q] < ht + 2^q$

If (a) occurs, then $MW[2 * a, j, \log_2 n] \geq n$. This can only happen if $i = j = 0$ and all image values are 1. In this case we get $M[0,0] = n$. If (b) occurs then the rectangle of height ht constructed in the forward pass ends at row n and cannot be enlarged further. In case (c), the forward pass is unable to attach the rectangle of height 2^q with top left corner at $[a * 2^q, j]$ to the current rectangle. A backward left to right pass from coulmn q is initiated. During this pass, the height of the rectangle is increased if possible. An example is given in Figure 10. The column

```
The following steps are done for all $[i,j]$, $0 \le i,j < N$.
Step 1   { Identify 1's that are not followed by a 1 }
        if $I[i,j] = 1$ and $(j = n-1$ or $I[i,j+1] = 0)$
        then $Z[i,j] := 1$
        else $Z[i,j] := 0$
Step 2   { Prefix sum $Z$ by rows }
        $P[i,j] := \sum_{k<j} Z[i,k]$
Step 3   { Table of last 1's }
        if $Z[i,j] = 1$ then $LAST[i,P[i,j]] := j$
Step 4   { Compute $R$ }
        if $I[i,j] = 0$ then $R[i,j] = 0$
        else begin
            $k :=$ least $l$ such that $LAST[i,j] \ge j$
            $R[i,j] := l-j+1$
          end
```

Figure 5 Algorithm to compute R

labeled $R(i,j)$ gives the R values for one column j of a 16×16 image. Once again, it is easy to see that M can be computed in $O(logn)$ time by an n^2 processor CREW PRAM.

4 Hypercube Algorithm For MAT

Z	0	0	1	0	0	0	0	0
P	0	0	0	1	1	1	1	1
LAST	2							
R	3	2	1	0	0	0	0	0
Z	0	0	0	0	1	0	0	0
P	0	0	0	0	0	1	1	1
LAST	4							
R	5	4	3	2	1	0	0	0
Z	0	0	0	0	0	0	1	0
P	0	0	0	0	0	0	0	1
LAST	6							
R	0	0	5	4	3	2	1	0
Z	0	0	0	0	0	0	0	1
P	0	0	0	0	0	0	0	0
LAST	7							
R	0	0	0	5	4	3	2	1
Z	0	0	0	0	0	0	0	1
P	0	0	0	0	0	0	0	0
LAST	7							
R	0	7	6	5	4	3	2	1
Z	0	0	0	0	0	0	0	1
P	0	0	0	0	0	1	1	1
LAST	4	7						
R	0	4	3	2	1	0	2	1
Z	0	0	0	0	1	0	0	0
P	0	0	0	0	0	1	1	1
LAST	4							
R	0	0	3	2	1	0	0	0
Z	0	0	0	0	0	1	0	0
P	0	0	0	0	0	0	1	1
LAST	5							
R	0	0	4	3	2	1	0	0

Figure 6 The value *Z*, *P*, *LAST* and *R* of image in Figure 1

```
Step 1   { Compute $MaxWidth$ }
      $MaxWidth[i,j,0] = R[i,j]$
      for $k = 1$ to $\log_2 n$ do
        $MaxWidth[i,j,k] = \min\{MaxWidth[i,j,k-1], MaxWidth[i+2^{k-1}, j, k-1]\}$
Step 2   { Compute $M$ }
      $ht := 0;$ $width := \infty$ $a := i;$ $r := \log_2 n;$
      while $r \geq 0$ do
      begin
        $l :=$ largest $q \leq r$ such that $MaxWidth[a,j,q] \geq ht + 2^q$
           (set $l$ to null if there is no such $q$)
        if $l$ = null then $r = -1$
             else begin
                 $ht := ht + 2^l;$
                 $width := \min\{width, MaxWidth[a,j,l]\}$
                 $a := a + 2^l;$
                 $r := l-1;$
               end;
      end;
      $M[i,j] = ht$
```

Figure 7 $O(n^2 logn)$ space computation of M

4.1 Terminology and Notation

We assume that $n = 2^k$ is a power of 2 and that a hypercube with n^2 processors is available. We assume a two dimensional indexing scheme of the porcessors in the hypercube [RANK90]. Thus PE $[i,j]$ refers to the PE in position $[i,j]$ of this scheme, $0 \leq i,j < n$ and $A(i,j)$ refers to variable A in $PE\ [i,j]$. The notation i_b refers to bit b of the binary representation of i. Thus, the binary representation of i is $i_{k-1} i_{k-2} .... i_0$. $i^{(b)}$ refers to the number whose

k	4	3	2	1	0
	0	0	0	0	0
					1
				1	2
					3
			1	2	4
					5
				3	6
					7
		1	2	4	8
					9
				5	10
					11
			3	6	12
					13
				7	14
					15

Figure 8 $MW[q,j,k]$ entries for column j of a 16×16 image
Numbers in boxes are the q values

binary representation is $i_{k-1}i_{k-2}....i_{b+1}\overline{i}_b i_{b-1}..i_0$ where $\overline{i}_b$ is the complement of i_b. By definition of a hypercube, $PE\ [i,j]$ and $PE\ [l,m]$ are neighbors (or adjacent) iff either $i = l$ and the binary representations of j and m differ in exactly one bit or $j = m$ and the binary representations of i and l differ in exactly one bit. In an SIMD hypercube, data can be transferred along a single dimension of the hypercube in unit time. The data transfer

$$A(i,j^{(b)}) \leftarrow B(i,j)$$

denotes a transfer from the B variable of $PE[i,j]$ to the A variable of $PE\ [i,j^{(b)}]$. Since the two processors are adjacent, the transfer takes 1 route. Actually, the above data transfer takes place

```
Step 1    { Compute $MW$ }
          $MW[q,j,0] := R[q,j],\ 0 \le q,\ j < n$
          for $k = 1$ to $\log_2 n$ do   { for $0 \le q < n/2^k$ and all $j$ }
           $MW[q,j,k] := \min\{MW[2*q,j,k-1],\ MW[2*q+1,j,k-1]\}$;
Step 2    { Compute M }
          $ht := 0;\ width := \infty;\ a := i;\ q := 0;$
          { Forward pass }
          while ( $q \le \log_2 n$) and ($a < \lfloor n/2^q \rfloor$) and ($MW[a,j,q] \ge ht + 2^q$)  do
          begin
            if $a$ is odd then
            begin
              $ht := ht + 2^q;$
              $width := \min\{width, MW[a,j,q]\}$;
            end;
            $q := q+1$;
            $a := \lceil a/2 \rceil$;
          end;
          case
          : $q > \log_2 n$:  $ht := n$;
          : $a < \lfloor n/2^q \rfloor$:
              while  $q > 0$ do  { Backward pass }
              begin
                $q := q-1$;
                $a := 2*a$;
                 if $MW[a,j,q] \ge ht + 2^q$ then
                 begin
                  $ht := ht + 2^q$;
                  $width := \min\{width, MW[a,j,q]\}$;
                  $a := a + 1$;
```

```
        **end;**
      **end;**
    $M[i,j] := ht;$
```

Figure 9 $O(n^2)$ space algorithm for M

for every $PE\ [i,j]$ in one route. We can limit the PEs involved in a route by providing a selectivity function as in:

$$A(i,j^{(b)}) \leftarrow B(i,j),\ \ (j_0 = 0)$$

Now the B variable value of $PE\ [i,j]$ for all i and j with $j_0 = 0$ is transferred to the A variable of $PE\ [i,j^{(b)}]$ for some fixed value of b.

4.2 Computation of R

We assume that the image matrix is initially distributed over the I variables of the n^2 processors. Specifically, $I(i,j)$ (i.e. the I variable of $PE\ [i,j]$) contains $I[i,j]$, $0 \le i,j < n$. The steps in the computation of R are given in Figure 11. Step 1 identifies the rightmost 1 in every sequence of 1's in every row of I. If $I(i,j) = 0$, it is not the location of a right most 1 and $A(i,j)$ is set to nil. If $I(i,j) = 1$ then it is a rightmost 1 if and only if either $j = n-1$ or $I(i,j+1) = 0$. This condition is checked by shifting the image values one position left on each row. In case the 1 at $[i,j]$ is a rightmost 1, $A(i,j)$ is set to j, the column address of this 1. Otherwise, $A(i,j)$ is set to nil. Since a row shift can be done in $O(logn)$ time on an $n \times n$ SIMD hypercube [RANK90], the time needed for Step 1 is $O(\log n)$. Lines 1-3 of Figure 12 gives one row of a 16×16 image matrix and the corresponding B and A values. Missing entries are nil.

Step 2 propagates the location of the rightmost one in each sequence of 1's to the left. This is done upto (but not including) the location of the nearest rightmost 1 on its left. The result, for our example, is shown in the last line of Figure 12. This propagation is accomplished by first considering subhypercubes of size 2, then subhypercubes of size 4, then 8,...., and finally n. In iteration b of Step 2, subhypercubes of size 2^{b+1} are considered. Each such subhypercube

$MW(q,j,k)$				$R(i,j)$	Forward			Backward			$M(i,j)$
k=4	k=3	k=2	k=1		q=**0**	q=**1**	q=**2**	q=**2**	q=**1**	q=**0**	
0	0	4	4	4	0	0	0	4	4	4	4
				5	1	3			3	3	3
			6	6	0	2			2	2	2
				7	1					1	1
		0	0	0							0
				0							0
			8	10	0	2			4	5	5
				8	1	1			3	4	4
	0	0	10	12	0	0			2	3	3
				10	1					2	2
			0	7	0					1	1
				0							0
		2	8	8	0	0			2	3	3
				8	1					2	2
			2	5	0					2	2
				2	1						1

Figure 10 Compute M from R by using forward and backward passes.
Entries under forward and backward are ht values.

```
Step 1   [ Identify rightmost 1's ]
        $B(i,j) := I(i,j)$;
        Shift B values by rows one column left and set $B(i,n-1)$ to 0;
        **if** $(I(i,j) = 1)$ **and** $(B(i,j) = 0)$
        **then** $A(i,j) = j$ { rightmost 1}
        **else** $A(i,j) := nil$;
        $C(i,j) := A(i,j)$;
Step 2   [Propagate rightmost 1's to the left ]
        **for** $b := 0$ **to** $k-1$ **do** {$n = 2^k$ }
        **begin**
         $D(i,j^{(b)}) \leftarrow C(i,j)$;
         $A(i,j) := D(i,j)$, $((A(i,j) = nil)$ and $j_b = 0)$;
         $C(i,j) := D(i,j)$, $((C(i,j) = nil)$ and $j_b = 0)$
                                **or** $(D(i,j) \neq nil$ and $j_b = 1)$;
        **end;**
Step 3   [ Compute R ]
        **if** $I(i,j) = 0$ **then** $R(i,j) := 0$
        **else** $R(i,j) := A(i,j) - j + 1$;
```

Figure 11 Hypercube computation of R

contains a left and a right subhypercube of size 2^b. The left subhypercube consists of PEs $[i,j]$ such that $j_b = 0$ while in the right subhypercube $j_b = 1$. At the start of each iteration the following is true in each subhypercube:

C1) The A value is either nil or is its correct final value

C2) All processors have the same C value and this is the column index (if any) to be

b		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	I	0	1	1	0	1	1	0	1	1	0	1	1	1	1	1	0
	B	1	1	0	1	1	0	1	1	0	1	1	1	1	1	0	0
	A			2			5			8						14	
	C			2			5			8						14	
	D				2	5					8						14
0	*A*			2		5	5			8						14	
	C			2	2	5	5			8	8					14	14
	D	2	2					5	5			8	8	14	14		
1	*A*	2	2	2		5	5			8				14	14	14	
	C	2	2	2	2	5	5	5	5	8	8	8	8	14	14	14	14
	D	5	5	5	5	2	2	2	2	14	14	14	14	8	8	8	8
2	*A*	2	2	2	5	5	5			8	14	14	14	14	14	14	
	C	2	2	2	2	2	2	2	2	8	8	8	8	8	8	8	8
	D	8	8	8	8	8	8	8	8	2	2	2	2	2	2	2	2
3	*A*	2	2	2	5	5	5	8	8	8	14	14	14	14	14	14	
	C	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
	R	0	2	1	0	2	1	0	2	1	0	5	4	3	2	1	0

Figure 12 Example hypercube computation of R

propagated to the size 2^b subhypercube on the left of this one.

It is easy to see that C1 and C2 are true at the start of the iteration with $b = 0$. Assume they are true at the start of the iteration with $b = q$. Each of the two subhypercubes in each size 2^{b+1} subhypercube sends its C values to the other subhypercube. This is retained in the D variables. If $A\ (i,j)$ is nil then PE $[i,j]$ doesn't have its final A value yet. Furthermore, if $j_b = 0$ then the current D value came from the right subhypercube of size 2^b. >From C2 it follows that this is the final A value (or nil) for PE $[i,j]$. If $j_b = 1$, the D value came from the left subhypercube and so cannot be the final A value. So, C1 is true at the start of the next iteration. As for C2, if a processor is in the left subhypercube (i.e., $j_b = 0$) and $C(i,j) \neq nil$ then its present $C(i,j)$ value is to be propagated to the left . If $C(i,j) = nil$ then the current $C(i,j)$ value in its right subhypercube is to be propagated. Hence the third line of the **for** loop of Step 2 ensures that C2 is true at start of the the next iteration. The truth of C1 and C2 imply the correctness of Step 2. The time complexity of Step 2 is easily seen to be $O(\log n)$. Step 3 is trivially correct and takes $O(1)$ time. Hence the overall complexity of our hypercube algorithm (Figure 11) to compute R is $O(\log n)$.

4.3 Computation of M

We shall show how the $O(n^2)$ space algorithm of Figure 9 can be run on an n^2 processor SIMD hypercube using $O(1)$ space per processor and $O(\log^2 n)$ time. To meet the $O(1)$ space requirement, the matrix MW computed in Step 1 of Figure 9 is distributed over the n^2 processors in the following way:

1. $MW[q,j,0]$ is just $R[q,j]$ and is simply kept in the R variable of PE $[q,j]$
2. $MW[q,j,k]$ for $k > 0$ is stored in the S variable of $PE[2^{k-1} + q * 2^k, j]$.

Figure 13 gives the storage scheme for column j of MW for a $16{\times}16$ image. This corresponds to Figure 8. S_i denotes the S varaibale in $PE\ [i,j]$ (i.e. $S(i,j)$) and R_i denotes the R variable in $PE\ [i,j]$.

The algorithmn to compute MW and save it in the S variables as in Figure 13 is given in Figure 14. The computation follows the binary tree pattern of Figure 15. This figure shows the computation for a fixed column j. The numbers inside the nodes gives the row index of the processor. Thus the nodes labeled i represent $PE\ [i,j]$, $0 \le i < n$. The edges with arrowheads represent interprocessor data routing. An upward arrow represents the routing of the A value of

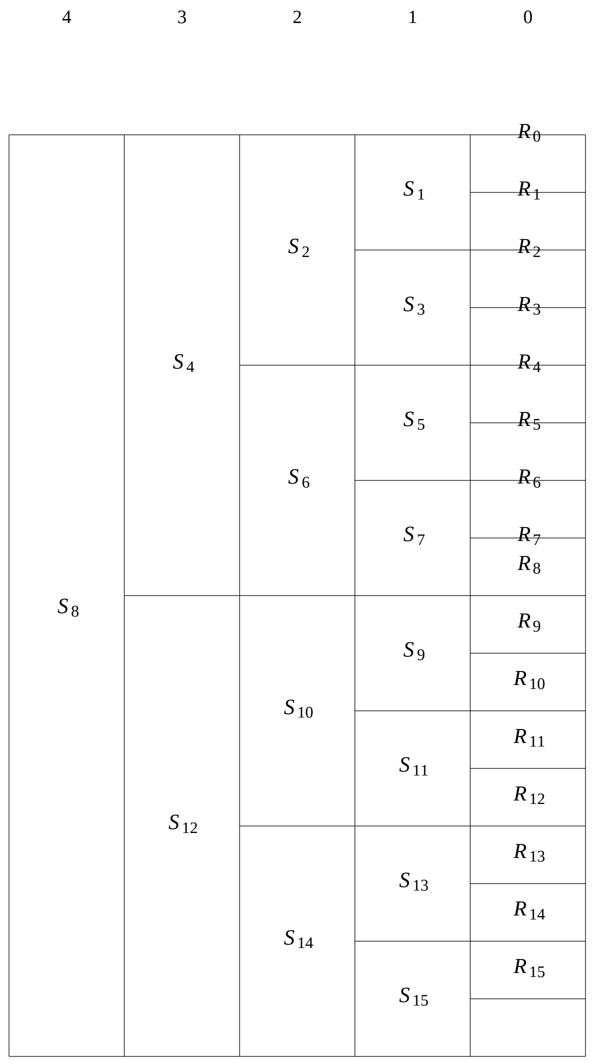

Figure 13 Distribution of MW

the lower processor to the B variable of the upper one while a downward arrow represents the routing of the A value of the upper processor to the S variable of the lower processor. The A variables are initialized to the R values as computed in Figure 11. On iteration k of the **for** loop the values of $MW[q,j,k+1]$ are computed. At the start of this iteration $A(i,j) = MW[i/2^k,j,k]$ for $i \ mod \ 2^k = 0$. For s such that $s \ mod \ 2^{k+1} = 0$, $MW(s/2^{k+1},j,k+1)$

$$= \min\{MW[2 * s/2^{k+1},j,k] \, , \, MW[2 * s/2^{k+1}+1,j,k]\}$$

$$= \min\{MW[s/2^k,j,k] \, , \, MW[s/2^k+1,j,k]\}$$

$$= \min\{A(s,j) \, , \, A(s+2^k,j)\}.$$

Since $s \ mod \ 2^{k+1} = 0$, $s_k = 0$ and $(s+2^k)_k = 1$. The first line of the **for** loop moves

$A(s+2^k,j)$ to $B(s,j)$. The second line computes $MW[s/2^{k+1},j,k+1]$ and saves it in $A(i,j)$ and the third line also saves it in $S(s+2^k,j)$. The complexity of the algorithm of Figure 14 is $O(\log n)$.

```
$A(i,j) := R(i,j)$;
**for** $k := 0$ **to** $\log_2 n - 1$ **do**
**begin**
    $B(i^{(k)},j) \leftarrow A(i,j), \quad (i_k = 1)$;
    $A(i,j) := \min\{A(i,j), B(i,j)\}, \quad (i_k = 0)$;
    $S(i^{(k)},j) \leftarrow A(i,j), \quad (i_k = 0)$;
**end**;
```

Figure 14 Hypercube computation of MW

In the forward and backward passes, each column of hypercube processors needs to read a subset of the S (or R) values stored in that column. Figure 16 gives the S values needed by the individual processors in column j. Again, S_i denotes $S(i,j)$. The q values correspond to the value of q in the first **while** loop of Step 2 of Figure 9. All of the values indicated in Figure 16 may not be needed as some processors will complete their **while** loop before $q = \log_2 n$. These processors will simply ignore the excess S values received.

Each column j processor can determine which S value is needed in each iteration of the **while** loop. Let $Z(i,j)$ be such that $PE\ [i,j]$ needs $S(Z(i,j),j)$ in the current iteration. All PEs can get their S values in $O(\log^2 n)$ time using the random access read (RAR) algorithm of [NASS81]. However, since $Z(i,j) \le Z(i+1,j)$ the sort steps used in the RAR algorithm may be eliminated and the RAR computed in $O(\log n)$ time.

Since $\log_2 n$ iterations of the **while** loop are made in the forward pass, the total time needed for the forward pass is $O(\log^2 n)$. In the backward pass, again, each processor can determine

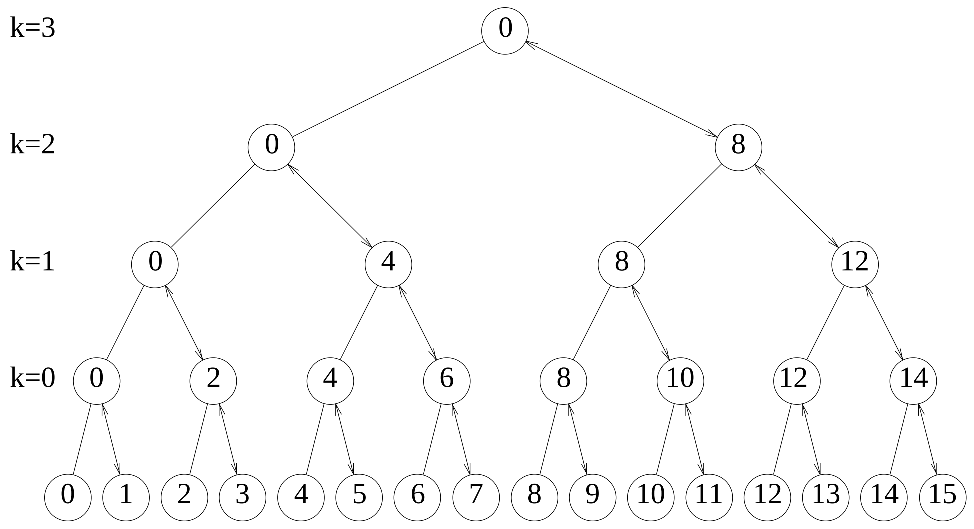

Figure 15 Computation of S(*,j) for n= 16

which S value it needs. Let $Z(i,j)$ be as above. Once again, $Z(i,j) \le Z(i+1,j)$ and the processors can get the needed S values in $O(\log n)$ time. Hence the total time needed for the backward pass is also $O(\log^2 n)$.

Once M has been computed. The T values can be obtained in an additional $O(\log n)$ time by performing shifts. Hence the MAT of an $n \times n$ image can be computed in $O(\log^2 n)$ time on an n^2 processor SIMD hypercube.

5 Area Of The OR Of Rectangles

The area of the OR (union) of n rectangles can be found on an SIMD hypercube with $2n^2$ processeors in $O(\log n)$ time. For this, we assume the $2n^2$ processors are logically arranged into a $2n \times n$ array with $PE[i,j]$ denoting the processor in position $[i,j]$, $0 \le i < 2n$, $0 \le j \le n$. The n rectangles are assumed to be isooriented (i.e., their sides are parallel to the x- and y- axes). Let $y_0, y_1, y_2, ..., y_k, y_0 < y_1 < .... < y_k$, be the distinct y coordinates of the horizontal (i.e., top and bottom) edges of the n rectangles. Clearly, $k < 2n$. The region bounded by the

PE[*,*j*],*q*	**0**	**1**	**2**	**3**	**4**
0	R_0	S_1	S_2	S_4	S_8
1	R_1	S_3	S_6	S_{12}	-
2	R_2	S_3	S_6	S_{12}	-
3	R_3	S_5	S_6	S_{12}	-
4	R_4	S_5	S_6	S_{12}	-
5	R_5	S_7	S_{10}	S_{12}	-
6	R_6	S_7	S_{10}	S_{12}	-
7	R_7	S_9	S_{10}	S_{12}	-
8	R_8	S_9	S_{10}	S_{12}	-
9	R_9	S_{11}	S_{14}	-	-
10	R_{10}	S_{11}	S_{14}	-	-
11	R_{11}	S_{13}	S_{14}	-	-
12	R_{12}	S_{13}	S_{14}	-	-
13	R_{13}	S_{15}	-	-	-
14	R_{14}	S_{15}	-	-	-
15	R_{15}	-	-	-	-

Figure 16 *R* and *S* values needed by column *j* processors in forward pass

horizontal lines with *y* coordinates y_i and y_{i+1} defines the i'th horizontal slab (or simply slab). Figure 17 shows an example with $n = 4$ and the corresponding horizontal slabs (S0 - S5). For this example, $k = 6$. The concept of a slab was also used in [LU88] and [CHAN87] to obtain the area. The steps in our area computation algorithm are:

Assume that the *n* rectangles are initially stored in the *n* PEs of row 0 of our $2n \times n$ array view of the hypercube. Each PE contains the top y, bottom y, left x, and right x coordinates of one rectangle. These are, respectively, in the variables *topy*, *bottomy*, *leftx*, and *rightx*. Step 1 is easily performed in $O(logn)$ time using $2n^2$ processors and the sorting algorithm of [NASS82]. Assume that the distinct *y* coordinates are left in $Y(i, 0)$, $0 \le i < k < 2n$ following the sort. The algorithm for Step 2 is given in Figure 18.

In Step 2.1, we sort the *n* rectangles by the key (*leftx*, *rightx*). This can be done in

Step 1 Obtain the y-coordinates of each slab.

Step 2 Compute the area in each slab.

Step 3 Add up the area contribution from the slab.

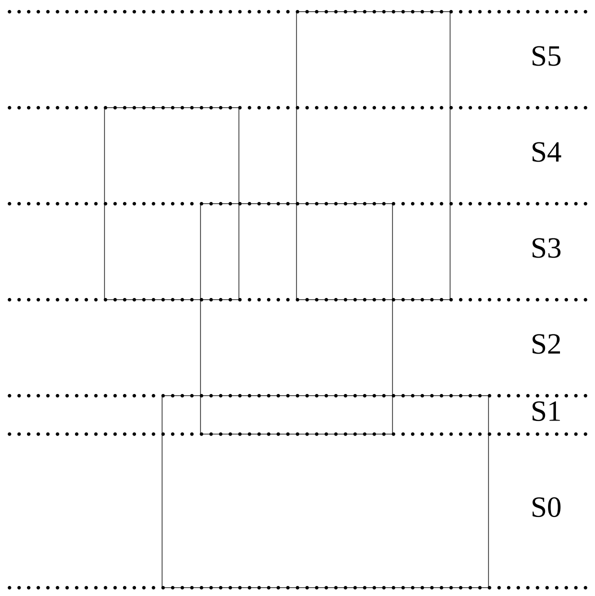

Figure 17 Slabs of iso-oriented rectangles

$O(logn)$ time using $2n^2$ processors [NASS82]. Step 2.2 broadcasts the sorted rectangles down the columns so that all processors in column j of the $2n \times n$ hypercube contain the *leftx* and *rightx* information of the j'th rectangle. Following Steps 2.3 and 2.4 the processors in row i

```
Step 2.1  Sort the rectangles into nondecreasing order of $(leftx,\ rightx)$.
          The result is in $PE\ [0,j],\ 0 \le j < n$.
Step 2.2  $PE\ [0,j]$ broadcasts its $(leftx,\ rightx,\ topy,\ bottomy)$ values to
          $PE\ [i,j],\ 0 \le i < 2n\ ,\ 0 \le j < n$.
Step 2.3  $PE\ [i,0]$ broadcasts its $Y$ value to $PE\ [i,j],\ 0 \le j < 2n,\ 0 \le i < k$.
Step 2.4  $Ybot(i,j) := Y(i,j),\ 0 \le i < k$
          Shift $Ybot$ up the columns by 1
Step 2.5  if rectangle in $PE\ [i,j]$ overlapes the slab in $PE\ [i,j]$ then
          set $contribute(i,j) := true$ and $RX(i,j) := rightx(i,j)$
          else set $contribute(i,j) := false$ and $RX(i,j) := 0$
Step 2.6  for $b := 0$ to $\log_2 n - 1$ do
          begin
          $NRX(i,j^{(b)}) \leftarrow RX(i,j)$
          $RX(i,j) := \max\{RX(i,j),\ NRX(i,j)\}$
            for processors $[i,j]$ with $j_b = 1$ do
               if $NRX(i,j) \ge rightx(i,j)$
               then { containment } $contribute(i,j) := false$
               else if $NRX(i,j) > leftx(i,j)$
                    then $leftx(i,j) := NRX(i,j)$;
          end;
Step 2.7  if $contribute(i,j)$
          then $area(i,j) := (Y(i,j) - Ybot(i,j) * (rightx(i,j) - leftx(i,j))$
          else $area(i,j) := 0$;
Step 2.8  Compute $area(i,0) := \sum_j area(i,j)$
```

Figure 18 Hypercube algorithm to obtain area in a slab.

know the y coordinates of the upper (Y) and lower ($Ybot$) horizontal lines that define the boundaries of the i'th slab. Step 2.5 determines if the j'th rectangle makes a possible area

contribution to the i'th slab. This is so only if

$bottomy(i,j) < Y(i,j)$ and $topy(i,j) > Ybot(i,j)$. In this case, this rectangle may make a contribution of up to $(rightx(i,j) - leftx(i,j)) * (Y(i,j) - Ybot(i,j)$ to the area. So, $contribute(i,j)$ is set to true. $RX(i,j)$ gives the right x coordinate of the rectangle overlap (if any). Following Step 2.5 each processor in row i with $contribute(i,j)$ = true defines a rectangle with height equal to the slab height and left and right x- coordianates given by $leftx(i,j)$ and $rightx(i,j)$. Some of these rectangles may overlap and we need to eliminate this overlap. This is accomplished in Step 2.6 by working in subhypercubes of size 2^{b+1}, $0 \le b < \log_2 n$. Each subhypercube of size 2^{b+1} consists of two subhypercubes of size 2^b that differ on bit b of their column index. $RX(i,j)$, at the start of iteration b, gives the maximum x coordinate of any of the rectangles in the subhypercube of size 2^b that contains $PE[i,j]$. Line 2 of the **for** loop ensures this is true on the next iteration and the initialization of Step 2.5 ensures this is true at the start of the iteration with $b = 0$. The **if** statement in the **for** loop is executed only by processors $[i,j]$ with $j_b = 1$ (i.e., the processor in the right size 2^b subhypercube of each size 2^{b+1} subhypercube). These processors determine if there is an overlap between their rectangle and any rectangle in the left 2^b subhypercube. Such an overlap is possible only with the rectangle that extends farthest to the right, i.e., to $NRX(i,j)$. If $NRX(i,j) \ge rightx(i,j)$ then the rectangle on the left totally covers the one in $PE[i,j]$. So, $PE[i,j]$ doesn't contribute to the area. Otherwise, if $NRX(i,j) > leftx(i,j)$ then the rightmost rectangle in the left subhypercube overlaps the rectangle in $PE[i,j]$ upto the x-coordinate $NRX(i,j)$. So, the new area contribution is only from $NRX(i,j)$ to $rightx(i,j)$. Hence $leftx(i,j)$ is updated to $NRX(i,j)$. One may verify that following iteration b of the **for** loop the rectangles in the row i processors with $contribute$ = true are disjoint. Step 2.7 computes the area contribution of each processor and Step 2.8 adds this up for each slab. Each of the steps of Figure 18 can be done in $O(logn)$ time [RANK90]. Step 3 of the overall algorithm for area simply requires us to compute $area(0,0) = \sum_i area(i,0)$. This is easily done in $O(logn)$ time.

6 Perimeter Of The OR Of Rectangles

The perimeter is computed in two stages. First the contribution of the vertical segments is obtained and then the contribution of the horizontal segments is obtained. The sum is the perimeter. The algorithm for the second stage is the same as that for the first except that the input rectangles are rotated by 90^o. To get the vertical contribution to the perimeter we modify the algorithm of Figure 18 slightly. Steps 2.6 through 2.8 are replaced by Step 2.6’ through Step 2.8’ (Figure 19). By summing $ht(i, 0)$ over all i one obtains the contribution of the vertical segment to the perimeter. The correctness of the resulting algorithm is easily established. Its complexity is $O(\log n)$.

```
Step 2.6'  for $b := 0$ to $\log_2 n-1$ do
             begin
              $NRX(i,j^{(b)}) \leftarrow RX(i,j)$
              $RX(i,j) := \max\{RX(i,j), NRX(i,j)\}$
              for processors $[i,j]$ with $j_b = 1$ do
                 if $NRX(i,j) \geq leftx(i,j)$ then $contribute(i,j) := false$;
             end;
Step 2.7'  if $contribute(i,j)$ then $ht(i,j) := 2 * (Y(i,j) - Ybot(i,j))$;
                                else $ht(i,j) := 0$
Step 2.8'  Compute $ht(i, 0) := \sum_j ht(i,j)$
```

Figure 19 Hypercube algorithm to obtain vertical perimeter contribution in each slab.

7 Conclusions

We have develop an $O(n^2)$ time serial algorithm to obtain the MAT of an $n \times n$ image. This is an improvement over the $O(kn^2)$ algorithm (k is the height of the largest square in the MAT) of [VO82]. Two CREW PRAM algorithms for the MAT were also developed . Both use n^2 processors and have a time complexity of $O(logn)$. One required $O(n^2 \log n)$ space while the other required only $O(n^2)$ space. Our hypercube algorithm for the MAT uses n^2 processors and has a time complexity of $O(\log^2 n)$ and requires $O(1)$ space per processor. We also presented an $O(logn)$ time hypercube algorithm to obtain the area of the OR of n rectangles and another algorithm with same asymptotic complexity to obtain the perimeter of the OR of n rectangles.

8 References

[BLUM67] H. Blum, *Models for the perception of speech and visual form*, MIT press, 1967, pp 362-380.

[GARE79] M. R. Garey and D. S. Johnson, *Computers and Intractability*, W. H. Freeman and Company, San Francisco, 1979.

[CHAN87] S. Chandran and D. Mount, "Shared memory algorithms and the medial axis transform", IEEE 1987 Workshop on CAPAMI, pp 44-50.

[GUTI84] R. Guting, "An optimal contour algorithm for iso-oriented rectangles", Journal of Algorithms, 5, 1984, pp 303-326.

[JARV90] P. Javis, *Parallel Solutions For some Design Automation Problems, PhD Dissertation, University of Minnesota.*

[KRUS85] C. Kruskal, L. Rudolph, and M. Snir, "The power of parallel prefix", IEEE International Conference on Parallel Processing, 1985, pp 180-185.

[LEE82] D. T. Lee, "Medial axis transformation of a planar shape", IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI 4, No. 4, July 1982, pp 363-369.

[LU88] M. Lu and P. Varman, "Optimal algorithms for rectangle problems on a mesh-connected computer", Journal of Parallel and Distributed Computing 5, 1988, pp 154-171.

[NASS81] D. Nassimi and S. Sahni, "Data broadcasting in SIMD Computers", IEEE Transactions on Computers, No. 2, Vol. C-30, 1981, pp 101-107.

[NASS82] D. Nassimi and S. Sahni, "Optimal BPC permutations on a cube connected computer", IEEE Transactions on Computers, No. 4, Vol. C-31, 1982, pp 338-341.

[RANK90] S. Ranka and S. Sahni, *Hypercube algorithms with applications to image processing and pattern recognition*, Springer-Verlag, 1990.

[ROSE82] A. Rosenfeld and A. C. Kak, *Digital Picture Processing*, Academic Press, New

York, 1982.

[V082] K. Vo, "Prob 80-4", Journal of Algorithms, 4(3), pp 366-368, 1982.

[WOOD84] D. Wood, "The contour problem for rectilinear polygons", Information Processing Letters, 17, 1984, pp 229-235.

[WU86] A. Y. Wu, S. K. Bhaskar and A. Rosenfeld, "Computation of geometric properties from the medial axis transform in $O(NlogN)$ time, Computer Vision, Graphics, and Image Processing, 34, 1986, pp 76-92.

[WU88] A. Y. Wu, S. K. Bhaskar and A. Rosenfeld, "Parallel computation of geometric properties from the medial axis transform", Computer Vision, Graphics, and Image Processing, 41, 1988, pp 323-332.