Reconfigurable Mesh Algorithms For The Area And Perimeter Of Image Components And Histogramming*

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

Abstract
We consider the following image processing problems: compute the area and perimeter of the components of an image, compute the histogram of an image, and histogram modification. Parallel reconfigurable mesh computer algorithms are developed for these problems. The area and perimeter of the components of an $N \times N$ image are computed in $O(logN)$ time by an $N \times N$ RMESH; the histogram of an $N \times N$ image is computed by an $N \times N$ RMESH in $O(\min\{\sqrt{B}\log(N/\sqrt{B}), N\})$ time where B is the number of gray scale values; and histogram modification is done in $O(\sqrt{N})$ time by an $N \times N$ RMESH.

Keywords and Phrases
reconfigurable mesh computer, parallel algorithms, image processing, area and perimeter of image components, histogram.

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

1 Introduction

Miller, Prasanna Kumar, Resis and Stout [MILL88abc] have proposed a variant of a mesh connected parallel computer. This variant, called a reconfigurable mesh with buses (RMESH), employs a reconfigurable bus to connect together all processors. Figure 1 shows a 4×4 RMESH. By opening some of the switches, the bus may be reconfigured into smaller buses that connect only a subset of the processors.

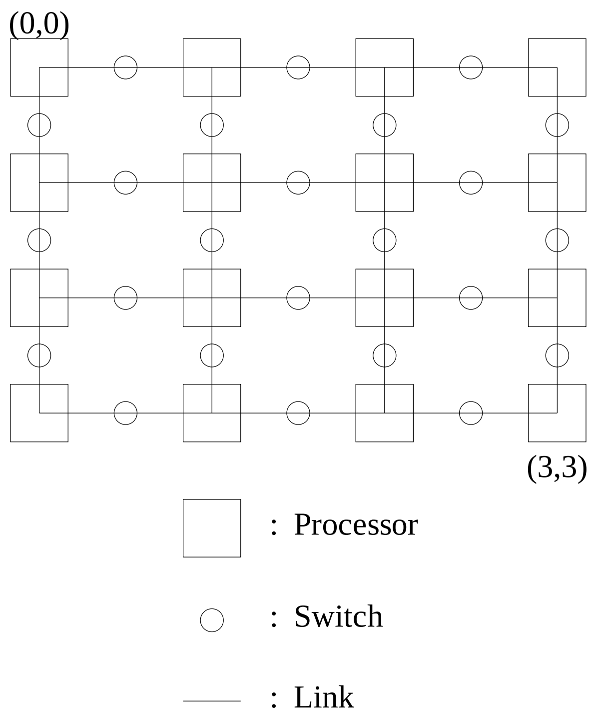


Figure 1 4×4 RMESH

The important features of an RMESH are [MILL88abc]:

1 An $N \times M$ RMESH is a 2-dimensional mesh connected array of processing elements (PEs). Each PE in the RMESH is connected to a broadcast bus which is itself constructed as an $N \times M$ grid. The PEs are connected to the bus at the intersections of the grid. Each processor has up to four bus switches (Figure 1) that are software controlled and that can be used to reconfigure the bus into subbuses. The ID of each PE is a pair (i,j) where i is the row index

and j is the column index. The ID of the upper left corner PE is (0,0) and that of the lower right one is $(N-1, M-1)$.

2 The up to four switches associated with a PE are labeled E (east), W (west), S (south) and N (north). Notice that the east (west, north, south) switch of a PE is also the west (east, south, north) switch of the PE (if any) on its right (left, top, bottom). Two PEs can simultaneously set (connect, close) or unset (disconnect, open) a particular switch as long as the settings do not conflict. The broadcast bus can be subdivided into subbuses by opening (disconnecting) some of the switches.

3 Only one processor can put data onto a given sub bus at any time

4 In unit time, data put on a subbus can be read by every PE connected to it. If a PE is to broadcast a value in register I to all of the PEs on its subbus, then it uses the command broadcast(I).

5 To read the content of the broadcast bus into a register R the statement R := content(bus) is used.

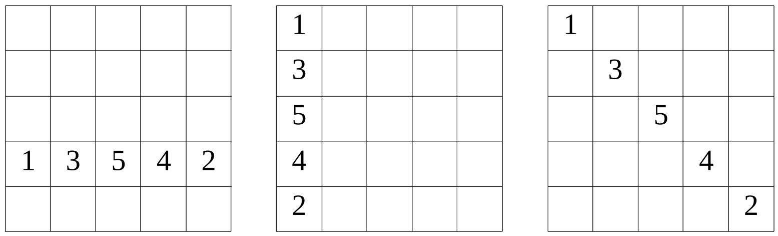
6 Row buses are formed if each processor disconnects (opens) its S switch and connects (closes) its E switch. Column buses are formed by disconnecting the E switches and connecting the S switches.

7 Diagonalize a row (column) of elements is a command to move the specific row (column) elements to the diagonal position of a specified window which contains that row (column). This is illustrated in Figure 2.

In this paper, we develop RMESH algorithms for the following image processing problems: compute the area and perimeter of the components of an image (Section 3), and histogram computation and modification (Section 4). First in Section 2, we develop RMESH algorithms for some basic data manipulation problems. These are used, later, in the development of our image processing algorithms.

2 Basic Data Manipulation Operations

In this section we define several data manipulation algorithms for RMESH multicomputers. These are used in later sections to develop algorithms for the image processing applications we consider.



(a) 4th row (b) 1st column (c) diagonalize

Figure 2 Diagonalize 4th row or 1st column elements of a 5×5 window

2.1 Window Broadcast

The data to be broadcast is initially in the A variable of the PEs in the top left $w \times w$ submesh. These PEs have ID (0,0) .. $(w-1, w-1)$. The data is to tile the whole mesh in such a way that $A(i,j) = A(i \ mod \ w, j \ mod \ w)$ ($A(i,j)$ denotes register A of the PE with ID (i,j)). The algorithm for this is given in Figure 3. Its complexity is $O(w)$ and is independent of the size of the RMESH.

2.2 Prefix Sum

Assume that N^2 values $A_0, A_1, ..., A_{N^2-1}$ are initially distributed in the A variables of an $N \times N$ RMESH such that $A(i,j) = A_{iN+j}$, $0 \le i,j, < N$. PE (i,j) is to compute a value $Sum(i,j)$ such that

$$Sum(i,j) = \sum_{k=0}^{iN+j} A_k, \quad 0 \le i,j < N$$

An $O(logN)$ algorithm for this is given in [MILL88a].

2.3 Data Sum

Initially, each PE of the $N \times N$ RMESH has an A value. Each PE is to sum up the A values of all the N^2 PEs and put the result in its B variable. I.e., following the data sum operation we have :

$$B(i,j) = \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} A(k,l), \quad 0 \le i,j < N$$

---

procedure *WindowBroadcast*(A,w);

{ broadcast the A values in the upper left $w \times w$ submesh }

begin

  for $j := 0$ **to** $w-1$ **do** { broadcast column j of the submesh }

  begin

    diagonalize the A variables in column j of the $w \times w$ submesh so that

    $B(i,i) = A(i,j),\ 0 \le i < w$;

    set switches to form column buses;

    $PE(i,i)$ broadcasts its B value on column bus i, $0 \le i < w$;

    $B(k, k\ mod\ w) := content(bus),\ 0 \le k < N$;

    set switches to form row buses;

    $PE(k, k\ mod\ w)$ broadcasts its B value on its row bus, $0 \le k < N$;

    $A(k,i)$:= content(bus) for $i\ mod\ w = j$, and $0 \le k < N$;

  end;

end;

Figure 3 Window broadcast

This can be done in $O(logN)$ time by first performing a prefix sum [MILL88a] and then having PE $(N-1,N-1)$ broadcast $Sum(N-1,N-1)$ to the remaining PEs in the RMESH. For this, all switches can be closed.

2.4 Shift

Each PE has data in its A variable that is to be shifted to the B variable of a processor that is s, $s > 0$, units to the right but on the same row. Following the shift, we have

$$B(i,j) = \begin{cases} null & j < s \\ A(i,j-s), & j \ge s \end{cases}$$

A circular shift variant of the above shift requires

$$B(i,j) = A(i,\ (j-s)\ mod\ N)$$

Let us examine the first variant first. This can be done in $O(s)$ time by dividing the send and receive processor pairs $((i,\ j-s),\ (i,j))$ into $s+1$ equivalence classes as below:

$$class\ k\ = \{((i,j-s),\ (i,j))\mid\ (j-s)\ mod\ (s+1) = k\}$$

The send and receive pairs in each class can be connected by disjoint buses and so we can accomplish the shift of the data in the send processors of each class in $O(1)$ time. In $O(s)$ time all the classes can be handled. The algorithm is given in Figure 4. The number of broadcasts is $s+1$. The procedure is easily extended to handle the case of left shifts. Assume that $s < 0$ denotes a left shift by s units on the same row. This can also be done with $s+1$ broadcasts.

---

procedure $Shift(s,A,B)$

{ Shift from $A(i,j)$ to $B(i,j+s)$, $s > 0$ }

begin

    All PEs disconnect their N and S switches;

    for k := 0 **to** s **do** { shift class k }

    begin

        $PE\ (i,j)$ disconnects its E switch if $(j-s)\ mod\ (s+1) = k$;

        $PE\ (i,j)$ disconnects its W switch and broadcasts

                $A(i,j)$ if $j\ mod\ (s+1) = k$;

        $B(i,j)$:= content(bus) for every PE (i,j) with $(j-s)\ mod\ (s+1) = k$;

    end;

end;

Figure 4 Shifting by $s, s > 0$

A circular shift of s can be done in $O(s)$ time by first performing an ordinary shift of s and then shifting $A(i,N-s),...,A(i,N-1)$ left by $N-s$. The latter shift can be done by first shifting $A(i,N-s)$, then $A(i,N-s+1)$,..., and finally $A(i,N-1)$. The exact number of broadcasts is $2s+1$.

Circular shifts of s, $s > N/2$ can be accomplished more efficiently by performing a shift of $-(N-s)$ instead. For $s \le N/2$, we observe that data from *PEs* $(i, 0)$, $(i, 1)$, $\cdots$ $(i, s-1)$ need to be sent to PEs (i,s), $(i,s+1)$, $\cdots$., $(i, 2s-1)$, resepectively. So, by limiting the data movement to within rows, s pieces of data need to use the bus segment between *PE* $(i,s-1)$ and (i,s). This takes $O(s)$ time. If only the data on one row of the $N \times N$ RMESH is to be shifted, the shifting can be done in $O(1)$ time by using each row to shift one of the elements. The circular shift operation can be extended to shift in $1 \times W$ row windows or $W \times 1$ column windows. Let *RowCircularShift* (A,s,W) and *ColumnCircularShift* (A,s,W), respectively, be procedures that shift the A values by s units in windows of size $1 \times W$ and $W \times 1$. Let A^{in} and A^f, respectively, denote the initial and final values of A. Then, for *ColumnCircularShift* we have

$$A^{in}(i,j) = A^f(q,j)$$

where PEs (i,j) and (q,j) are, respectively, the $a = i \bmod W$'th and $b = q \bmod W$'th PEs in the same $W \times 1$ column window and $b = (a-s) \bmod W$. The strategy of Figure 4 is easily extended so that *RowCircularShift* and *ColumnCircularShift* are done using $2s + 1$ broadcasts.

2.5 Consecutive Sum

Assume that an $N \times N$ RMESH is tiled by $1 \times M$ blocks (M divides N) in a natural manner with no blocks overlapping. So, processor (i,j) is the $j \bmod M$'th processor in its block. Each processor (i,j) of the RMESH has an array $X[0..M-1](i,j)$ of values. If $j \bmod M = q$, then PE (i,j) is to compute $S(i,j)$ such that

$$S(i,j) = \sum_{r=0}^{M-1} X[q](i, (j \text{ div } M) * M + r)$$

That is, the q'th processor in each block sums the q'th X value of the processors in its block. The consecutive sum operation is performed by having each PE in a $1 \times M$ block initiate a token that will accumulate the desired sum for the processor to its right and in its block. More specifically, the token generated by the q'th PE in a block will compute the sum for the $(q+1) \bmod M$'th PE in the block, $0 \le q < M$. The tokens are shifted left circularly within their $1 \times M$ block until each token has visited each PE in its block and arrived at its destination PE. The algorithm is given in Figure 5. The number of broadcasts is $3M-3$ as each row circular shift of -1 takes 3 broadcasts.

```
procedure ConsecutiveSum (X,S,M);
{ Consecutive Sum of X in 1×M blocks }
begin
  S(i,j) := X[((j mod M)+1) mod M](i,j), 0 ≤ i,j < N;
  for k := 2 to M do
  begin
    {circularly shift S in 1×M blocks and add terms }
    RowCircularShift (S, M,-1)
    S(i,j) := S(i,j) + X[((j mod M)+k) mod M](i,j),0 ≤ i,j < N;
  end;
end;
```

Figure 5 Consecutive sums in $1 \times M$ blocks

2.6 Sorting

N^2 elements, one per processor, can be sorted in $O(N)$ time on an $N \times N$ RMESH by simulating the $O(N)$ sorting algorithm for ordinary mesh computers [NASS79]. That $O(N)$ is optimal for an RMESH can be seen by considering the amount of data that might need to cross the boundary between the left $N/2$ columns and the right $N/2$ columns. This is $N^2/2$ in the worst case. The bandwidth of this boundary is N. Hence $O(N)$ time is needed to accomplish this data transfer. Miller et al. [MILL88a] present an $O(\log N)$ sorting algorithm for the case when N elements are to be sorted on an RMESH with N^2 processors. The initial and final configuration has the data in row 0 of the $N \times N$ RMESH.

2.7 RAR And RAW

The random access read (RAR) and random access write (RAW) operations are defined in [NASS81]. In a RAR each PE has a read address associated with it. This is the address of the PE whose A variable it wishes to read. In a RAW each PE has a write address which is the address of the PE to which it wishes to send the value of its A variable. Conflicts may be resolved arbitrarily. Miller et al. [MILL88a] have developed RMESH algorithms for RARs and RAWs. When k data items are to be moved in the RAR or RAW, their algorithm takes $O(\sqrt{k} + \log N)$ time, $k \leq N^2$. If the number of source and destination processors in each $k \times k$ block of PEs is $O(k)$, $1 \leq k \leq N$ then their algorithm takes $O(logN)$ time.

3 Area And Perimeter Of Connected Components

Let $I[0..N-1, 0..N-1]$ be a binary image. Two pixels $[u,v]$ and $[w,x]$ are adjacent iff either $|u-w| = 1$ and $|v-x| = 0$ or $|u-w| = 0$ and $|v-x| = 1$. The transitive closure of this adjacency relation taken over the nonzero pixels of I partitions these nonzero pixels into equivalnece classes called *connected components*. In several applications [DUDA73] it is necessary to know the area and perimeter of each of these components (the area of a component is the number of pixels in it and the perimeter is the number of pixels on the component contour).

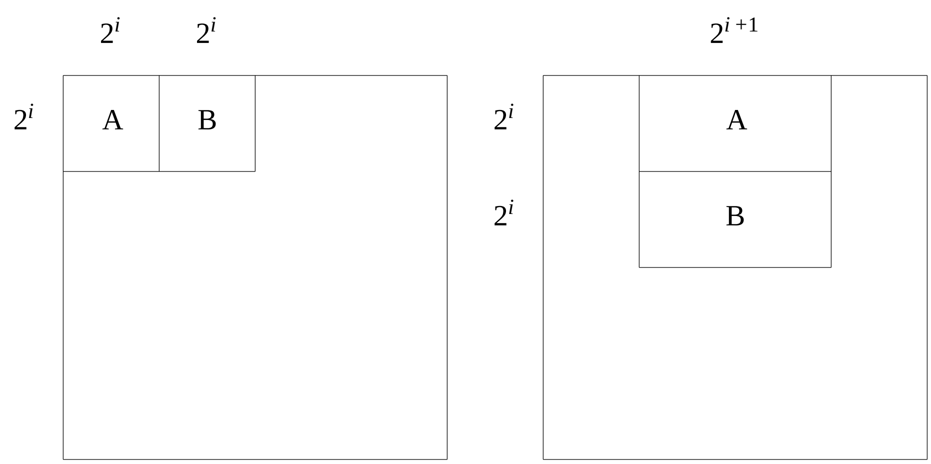
The initial configuration for the problems considered in this section is one in which each pixel is labeled by its component number. Specifically, each entry of I is a record with at least the two fields: *value* and *comp*. $I[i,j].value$ is a 0/1 pixel value and $I[i,j].comp$ gives the component to which this pixel belongs. If $I[i,j].value = 0$, then $I[i,j].comp = 0$. If $I[i,j].value = 1$, then $I[i,j].comp > 0$.

The area and perimeter of each component can be determined efficiently on hypercube and mesh connected computers by performing a sort. Consider the case of area determination. The pixels are first sorted by the field *comp*. Next the first and last pixel in each sequence with the same *comp* value is identified. The distance between these can be obtained by performing a data concentration [NASS81] of the ID of the processors containing the last pixel in each sequence. While the same technique can be applied to an RMESH, more efficient algorithms result from a different technique. On an $N \times N$ RMESH, the area and perimeter can be determined in $O(logN)$ time while it takes $O(N)$ time to sort N^2 elements.

3.1 Area

3.1.1 CRCW PRAM Algorithm

It is instructive to first consider a CRCW PRAM version of our algorithm. We assume, for simplicity, that N is a power of 2. Our algorithm employs the divide-and-conquer approach [HORO78]. Initially, we assume that each pixel is independent of the others; then we combine together blocks of pixels to obtain larger blocks. Two kinds of block combinations are performed. In one, we combine together two horizontally adjacent $2^i \times 2^i$ blocks as in Figure 6(a). In the other, two vertically adjacent $2^i \times 2^{i+1}$ blocks are combined as in Figure 6(b). Notice that when two horizontally adjacent blocks of size $2^i \times 2^i$ each are combined, we get a single block of size $2^i \times 2^{i+1}$. Further, when two vertically adjacent blocks of size $2^i \times 2^{i+1}$ are combined we get a block of size $2^{i+1} \times 2^{i+1}$. Beginning with $2^0 \times 2^0$ blocks, we alternately combine pairs of horizontal adjacent blocks and pairs of vertical adjacent blocks until only one $N \times N$ block remains. Consequently only the two kinds of block combinations described above are needed. Figure 7 shows the combination process for the case of a 4×4 image.



(a) Horizontally adjacent blocks (b) Vertically adjacent blocks

Figure 6 Block Adjacency

With each pixel $[i,j]$ we associate two additional fields: *update* and *area*. *update* is a

A	B	C	D
E	F	G	H
I	J	K	L
M	N	O	P

(a) Initial blocks

A B	C D
E F	G H
I J	K L
M N	O P

(b) Horizontal combination

A B E F	C D G H
I J M N	K L O P

(c) Vertical combination

A B C D E F G H
I J K L M N O P

(d) Horizontal combination

A - P

(c) Vertical combination

Figure 7 Block Combination

Boolean field and *area* is an integer field which will eventually be the number of nonzero pixels in the component $I[i,j].comp$. Initially, we have:

$$I[i,j].area = I[i,j].value, \quad 0 \le i,j < N$$

When two blocks are combined, the area fields of the boundary pixels are updated to correspond to the number of pixels in the new combined block that have the same *comp* value. Figure 8 gives an example. Figure 8(a) gives the *comp* values of the pixels. There are only two components in the figure. Figures 8 (b) - (f) give the *area* values. The blocks are as in Figure 7. In this small example, each pixel is always a boundary pixel. In larger examples, this is not so. Figure 9 shows the pixels that get updated when two blocks are combined. These are labeled "x". Following each combination, the following is true:

1	1	1	0
1	0	0	2
0	0	2	2
2	2	2	0

(a) *comp*.values

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

(b) $2^0 \times 2^0$ blocks

2	2	1	0
1	0	0	1
0	0	2	2
2	2	1	0

(c) $2^0 \times 2^1$ blocks

3	3	1	0
3	0	0	1
0	0	3	3
2	2	3	0

(d) $2^1 \times 2^1$ blocks

4	4	4	0
4	0	0	1
0	0	5	5
5	5	5	0

(b) $2^1 \times 2^2$ blocks

4	4	4	0
4	0	0	6
0	0	6	6
6	6	6	0

(b) $2^2 \times 2^2$ blocks

Figure 8 Updating *area* by block combination

I1: If $[i,j]$ is a boundary pixel of one of the two blocks just combined, then $I[i,j]$.*area* is the number of pixels in the new block with *comp* value equal to $I[i,j]$.*comp* unless

$I[i,j].comp = 0$.

In this latter case $I[i,j].area = 0$.

Consider the case of horizontal combination (Figure 9(a)). Assume that two $2^i \times 2^i$ blocks are being combined and that for every boundary pixel $[i,j]$ of each block, we have:

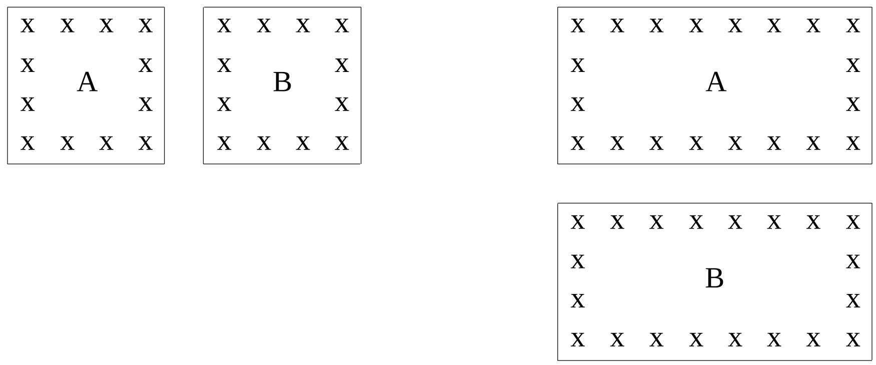
$I[i,j].area =$ number of pixels in the block with *comp* value equal to $I[i,j].comp$ unless $I[i,j].comp = 0$.

If $[i,j]$ is a boundary pixel in block $A(B)$ and $I[i,j].comp > 0$, then its *area* value changes iff there is a pixel on the boundary of block $B(A)$ with the same *comp* value. This follows from the definition of a connected component. If no pixel on the boundary of $B(A)$ has *comp* value $I[i,j].comp$, then no pixel in this block can have this *comp* value. Let $[u,v]$ be a pixel on the boundary of $B(A)$ such that $I[i,j].comp = I[u,v].comp \neq 0$. Then the updated *area* value for pixels $[i,j]$ and $[u,v]$ is $I[i,j].area + I[u,v].area$. Pairs $[i,j]$ and $[u,v]$ of matching pixels are found by dividing the boundary of each block into four lines: 2 horizontal and 2 vertical. Call these $top(x)$, $bottom(x)$, $left(x)$, $right(x)$, $x \in \{A,B\}$. Note that the lines are not disjoint. For example, $top(A)$ and $left(A)$ share one pixel (at the top left corner). All 16 combinations of lines from A and B are used to determine matching pairs. Each combination has the form $((Y(A),Z(B))$, $Y,Z \in \{top, bottom, left, right\}$. The code of Figures 10 and 11 describes how *area* is updated using a CRCW PRAM that has 2^{i+1} processors. For this to work correctly, it is necessary that the *area* values be read by all PEs before any PE attempts to write an *area* value. The complexity is $O(1)$. The code for the case of a vertical combination is the same. Since this combination has to be done $\log N$ times starting with blocks of size 1×1 and ending with a single block of size $N\times N$, the complexity of the procedure to compute area for boundary pixels is $O(logN)$.

Once we have combined blocks as described above then it is the case that the area of any component n is

$$\max\{I[i,j].area \mid I[i,j].comp = n\}$$

To get the condition where $I[i,j].area$ is the area of the component $I[i,j].comp$, $0 \le i,j < N$ we can run the block combination process backwards. The $N\times N$ block is decomposed into 2 , each of these is then decomposed into 2, and so on until we have N^2 1×1 blocks. At the start of each decomposition step, the boundary pixels contain the correct area value. The decomposition results in the correct area values in the boundary pixels of the decomposed blocks. The process



(a) Horizontal combination (b) Vertical combination

Figure 9 Boundary pixels labeled *x*

$I[i,j].update :=$ false, $0 \le i,j < N$

for *sideA* $\in$ {*top*, *bottom*, *left*, *right*} **do**

  for *sideB* $\in$ {*top*, *bottom*, *left*, *right* } **do**

    CombineLines(*sideA*,*sideB*);

Figure 10 Combine blocks *A* and *B*

is similar to that described earlier for block combination and we omit the details.

---

procedure *CombineLines* (*sideA*,*sideB*);

{update *area* for pixels on boundary lines *sideA* and *sideB* of blocks of *A* and *B* }

Let $|sideA|$ and $|sideB|$, respectively, be the number of pixels

on boundary line *sideA* of *A* and boundary line *sideB* of *B*;

PE (c,d) examines the *c*'th pixel, $0 \le c < |sideA|$ of *sideA* of *A*

and the *d*'th pixel, $0 \le d < |sideB|$ of *sideB* of *B*.

Let these pixels, respectively, be $[i,j]$ and $[u,v]$;

if $I[i,j].comp = I[u,v].comp$

then case

  $I[i,j].update$ **and not** $I[u,v].update$:

    $I[u,v].update := true$; $I[u,v].area := I[i,j].area$**;**

  not $I[i,j].update$ **and** $I[u,v].update$:

    $I[i,j].update := true$; $I[i,j].area := I[u,v].area$;

  not $I[i,j].update$ **and not** $I[u,v].update$**:**

    $I[i,j].update := true$; $I[u,v].update := true$;

    $I[i,j].area := I[i,j].area + I[u,v].area$;

    $I[u,v].area := I[i,j].area$;

  endcase;

end;

Figure 11 Combining two boundary lines

3.1.2 RMESH Algorithm

The RMESH algorithm works like the CRCW PRAM algorithm. We need to provide only the details for the code of Figure 11 (i.e., procedure CombineLines). Figure 12 gives the RMESH code for the case of horizontal combination. An $N \times N$ RMESH is assumed and PE (i,j) of the RMESH represents pixel $[i,j]$, $0 \le i,j < N$. The code for a vertical combination is similar. The complexity for both is $O(1)$. So, the complete area determination algorithm takes $O(logN)$ time.

```
procedure CombineLines(sideA,sideB);
{RMESH version }
  diagonalize the update, comp, and area values of sideB of block B and
  broadcast on row buses to all PEs on the same row in block A;
  the PEs of block A read their row buses and store the values read in
  variables updateB, compB, and areaB, respectively;
  diagonalize the update, comp, and area values of sideA of block A
  and broadcast on column buses to all PEs on the same column in block A;
  the PEs of block A read their column buses and store the values read in
  variables updateA, compA, and areaA;
  {now the PE in position [a,b] of block A has the information from the
  a'th pixel of sideA of A and b'th pixel of sideB of B}
  Each PE (a,b) of block A does the following:
  if compA = compB then
    case
      updateA and not updateB: updateB := true ;areaB := areaA;
      not updateA and updateB: updateA := true ;areaA := areaB;
      not updateA and not updateB : updateA := true ; updateB := true ;
            areaA := areaA + areaB; areaB := areaA ;
   endcase;
 { broadcast back to sideB}
  set up row buses in the AB combined block;
  every PE (a,b) of block A for which updateB(a,b) is true
  disconnects its W switch and broadcasts areaB;
  the diagonal PEs of block B read their buses and  if a value is read, this
  is broadcast to the appropriate PE of sideB using the reverse of a diagonalize,
  this PE in turn updates its areaB value and sets its update value to true;
 { broadcast to sideA }
  this is similar to that for sideB;
```

Figure 12 RMESH version of *CombineLines*

3.2 Perimeter

This can be done by preprocessing the image so that $I[i,j] = 1$ iff $[i,j]$ is a boundary pixel. This preprocessing is straightforward and requires each pixel to examine the pixels (if any) on its north, south, east, and west boundaries. Following the preprocessing, we see that the perimeter and area of a component are the same. Hence, the $O(logN)$ algorithm of the preceding section can be used.

4 Histogram

Let $I[0..N-1,0..N-1]$ be an $N \times N$ digitized image with $I[i,j]$ being the gray scale value of the pixel $[i,j]$. Let B be the range in gray scale values of the N^2 pixels. So, $0 \le I[i,j] < B$, $0 \le i,j < N$. The *histogram* of I is a vector H such that

$$H[a] = |\{[i,j] \mid I[i,j] = a,\ \ 0 \le i,j < N\}|$$

I.e., $H[a]$ is the number of pixels that have the gray value a. The histogram of an image has applications in image enhancement and segmentation.

Several parallel algorithms to compute the histogram have been proposed in the literature. Siegel et al. [SIEG81] develop an algorithm for a p processor, $p \le N^2$, PASM muticomputer while Yasrebi and Browne [YASR83] do so for the TRAC multicomputer. Grinberg, Nudd, and Etchells [GRIN84] consider the computation of H on an N^2 processor cellular machine called the 3-D machine. The resulting algorithm has time complexity $O(N)$. The histogram, H, is easily computed in $O(N^2)$ time on a serial single processor computer. This is optimal for such a computer as each of the N^2 pixels needs to be examined and such a computer can examine only one pixel at any time. On a parallel computer with N^2 processors, H can be computed in the time needed to sort N^2 elements. The strategy for this is given in Figure 13. The processors are indexed $0..N^2-1$ and the initial configuration has $I(iN+j) = I[i,j]$, $0 \le i,j < N$. The final configuration has the nonzero entries of H computed in $H(b).gray$ and $H(b).value$, $0 \le b < k$ for some $k \le \min\{N^2,B\}$. If $H(a).gray = q$, then $H[q] = H(a).value$.

Following the sort of step 1 the I values form a nondecreasing sequence. Step 2 identifies the end (tail) and start (head) of each subsequence of I that is comprised solely of equal gray values. The length of each such sequence is one of the nonzero entries in H. This length is easily computed from the addresses of the processors containing the head and tail of the subsequences. For this, the addresses are concentrated to the left in steps 3 and 4 and the necessary computation done in step 5. Since on all proposed parallel computer models data concentration

Step 1 Sort the $I(i)$, $0 \le i < N^2$.

Following the sort, we have $I(i) \le I(i+1)$, $0 \le i < N^2-1$;

{ assume that $I(-1) = -1$ and $I(N^2) = B$ }

Step 2 **if** $(i = N^2-1)$ **or** $(I(i) < I(i+1))$ **then** [$tail(i) := true$; $address1(i) := i$]

else $tail(i) := false$;

if $(i = 0)$ **or** $(I(i) > I(i-1))$ **then** [$head(i) := true$; $address2(i) := i$]

else $head(i) := false$;

$0 \le i < N^2$;

Step 3 concentrate $(I(i), address1(i))$ for $tail(i) = true$

to the left;

Step 4 concentrate $(address2(i))$ for $head(i) =$ true to the left;

Step 5 the processors i that received tuples in the concentration of steps 4

and 5 do the following :

$H(i).gray := I(i)$; $H(i).value := address1(i) - address2(i) + 1$

Figure 13 Computing H using a sort

takes less time than the sorting of M elements using M processors, the complexity of the algorithm of Figure 13 is determined by the complexity of the sort of step 1. Note also that since the head of one sequence is adjacent to the tail of the previous one, the concentration of step 4 can be replaced by a right shift by one of the *address* 1's of the tails. To get *address* 2, we need to add one to the just shifted *address* 1 values.

Tanimoto [TANI84] has developed an algorithm to compute the histogram on a pyramid machine. The complexity of this algorithm is $O(B + logN)$ on a pyramid with an $N \times N$ base. Bestul and Davis [BEST89] compute the histogram on an N^2 PE SIMD hypercube in $O(\sqrt{B} + \log(N/B))$ time. Their algorithm assumes $B \ll N$.

In Section 4.1 we show how to compute H on an N^2 processor RMESH in time less than that needed to sort N^2 elements on an $N \times N$ RMESH. This algorithm is for the case $B < N^2$. For

$B \geq N^2$, the algorithm of Figure 13 may be used to compute H in $O(N)$ time. Recall that N^2 elements can be sorted on an $N \times N$ RMESH in $O(N)$ time. In Section 4.2 we develop an algorithm to do histogram modification on an $N \times N$ RMESH.

4.1 Computing *H*

We shall develop two algorithms. One requires $O(\sqrt{B})$ memory per processor and the other requires $O(1)$. The first of these uses fewer broadcast steps than does the second.

4.1.1 $O(\sqrt{B})$ Memory Algorithm

First consider the case $B = N$. Initially, we have $I(i,j) = I[i,j]$, $0 \leq i,j < N$. On termination, the H values are stored in column 0 of the RMESH. I.e., $H(i, 0) = H[i]$, $0 \leq i < N$. Our strategy is to first have each $\sqrt{N} \times \sqrt{N}$ (assume for simplicity that N is a perfect square) sub RMESH work independently and compute H for the $\sqrt{N} \times \sqrt{N}$ portion of the image it contains. These $\sqrt{N} \times \sqrt{N}$ sub RMESHs are obtained by a natural $\sqrt{N} \times \sqrt{N}$ tiling of the $N \times N$ RMESH. The sub RMESHs can work independently by simply disconnecting their boundary switches. Figure 14 shows a 16×16 RMESH partitioned into 16 4×4 sub RMESHs. Processors with the same label are in the same sub RMESH. Each $\sqrt{N} \times \sqrt{N}$ sub RMESH has $B = N$ processors. Each of these is to compute $H[a]$ for exactly one value of a, $0 \leq a < B$. Of course, the H value computed is only for the image partition contained in the sub RMESH. Processor (i,j) of a $\sqrt{N} \times \sqrt{N}$ sub RMESH computes $H[i + j\sqrt{N}]$, $0 \leq i,j < \sqrt{N}$. Figure 15 shows the H values to be computed by each processor of a 4×4 sub RMESH.

In order to compute the H's as described, each processor (i,j) of a sub RMESH first computes an array $A[0..\sqrt{N}-1]$ of values where $A[b](i,j)$ is the number of pixels in row i of the sub RMESH that have gray value equal to $j\sqrt{N} + b$, $0 \leq b < \sqrt{N}$. This can be computed in $O(\sqrt{N})$ time by obtaining the contribution of each of the $\sqrt{N}$ columns in the sub RMESH in a different iteration. On iteration k (see Figure 16), column k of the sub RMESH broadcasts its I values along row buses that are confined to the sub RMESH. Each processor reads its row bus and determines if the I value read contributes to its A array. Note that the A array of the (i,j)'th PE of a sub RMESH accounts for gray values in the range $j\sqrt{N}$ through $(j+1)\sqrt{N}-1$. If so, the appropriate A value is to be updated. Let $T(i,j)$ be the I value read by the (i,j)'th processor of the sub RMESH. To contribute to the A array of this processor, T must be such that $j\sqrt{N} \leq T(i,j) < (j+1)\sqrt{N}$. If this is so, then $A[T(i,j) - j\sqrt{N}]$ corresponds to the gray value $T(i,j)$.

Once the A arrays have been computed, the H values as given in Figure 15 can be

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

Figure 14 Partitioning a 16×16 RMESH into 4×4 sub RMESHs

computed in $O(\sqrt{N})$ time by using the consecutive sum operation of Section 2. Following this, each processor of the $N \times N$ RMESH contains in H a partial histogram value. The situation for a 16×16 RMESH is shown in Figure 17. We need to add together all partial values that contribute to the same histogram value. This is done by first adding along columns. The N processors in each column contain partial values for exactly $\sqrt{N}$ different histogram values. In fact, the column j H values so far computed contribute towards $H[(j \bmod \sqrt{N})\sqrt{N}],\ldots,$ $H[(j \bmod \sqrt{N})\sqrt{N} + \sqrt{N}-1]$. For each column j, the H values that contribute to the same histogram value $H[a]$ are added together. Note that $(j \bmod \sqrt{N})\sqrt{N} \le a < (j \bmod \sqrt{N})\sqrt{N} + \sqrt{N}$ and that the H value in PE (i,j) contributes to $H[b]$ where $b = (j \bmod \sqrt{N})\sqrt{N} + i \bmod \sqrt{N}$. The

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

Figure 15 The $H[a]$ computed by each PE in a 4×4 sub RMESH

additions can be done in $\sqrt{N}\log N$ time by using the binary tree scheme of Figure 18. The addition at each node requires $\sqrt{N}$ values to be shifted. This takes $O(\sqrt{N})$ times. We require that the $\sqrt{N}$ sums computed for column j be left in the R variables of the processors (i,j) such that $(j \bmod \sqrt{N})\sqrt{N} \le i < (j \bmod \sqrt{N})\sqrt{N} + \sqrt{N}$. The scheme of Figure 18 saves the results in the top $\sqrt{N}$ processors of each column. The desired configuration for R can be obtained by doing a window broadcast down the columns and then zeroing out the R values in the processors that don't need these values.

To obtain the histogram values we now need to sum the R values on each row. This is easily done in $O(logN)$ time using the data sum operation. The overall complexity of our histogram algorithm for the case $N = B$ is therefore $O(\sqrt{N} logN)$.

Next, consider the case $B < N$. For simplicity, assume that $\sqrt{B}$ divides N. Instead of tiling the $N \times N$ mesh with $\sqrt{N} \times \sqrt{N}$ blocks as for the case when $B = N$, this time use $\sqrt{B} \times \sqrt{B}$ blocks and proceed as for the case $B = N$. The **for** loop of Step 3 of Figure 16 is now to be run only for k equal to 0 through $\sqrt{B}-1$, and each processor accumulates only $\sqrt{B}$ A values. The computation of A takes $O(\sqrt{B})$ time. The consecutive sum operation to get the partial histogram values also

{ computing A in a $\sqrt{N} \times \sqrt{N}$ sub RMESH }

{ i and j are PE indices relative to the sub RMESH }

Step 1 Every PE (i,j) initializes its A array to 0

$A[b](i,j) := 0,\ 0 \le b,i,j < \sqrt{N}$

Step 2 Set up row buses local to the sub RMESH;

Step 3 **for** $k := 0$ **to** $\sqrt{N}-1$ **do**

begin

all PEs on column k broadcast their I values on their row bus;

$T(i,j)$:= content(bus);

if $j\sqrt{N} \le T(i,j) < (j+1)\sqrt{N}$ {T is a value for PE (i,j)}

then $A[T(i,j)-j\sqrt{N}](i,j) := A[T(i,j)-j\sqrt{N}](i,j)+1$;

end;

Figuire 16 Computing A in a $\sqrt{N} \times \sqrt{N}$ sub RMESH

takes $O(\sqrt{B})$ time as it is confined to sub columns of size $\sqrt{B}$. Suming up partial histogram values along columns takes $O(\sqrt{B}\log(N/\sqrt{B}))$ time as the height of the summing tree of Figure 18 is $O(log(N/\sqrt{B}))$ and at each summation node $\sqrt{B}$ values are to be shifted. The results of the column sum process are left in rows 0 through $B-1$ of the RMESH. These rows need to perform a row sum operation. Since each row has $N/\sqrt{B}$ values to be summed, the time needed is $O(logN/\sqrt{B})$. The overall complexity becomes $O(\sqrt{B}\log(N/\sqrt{B}))$.

Next, suppose $N < B < N^2$. For simplicity, assume that $\sqrt{B}$ divides N. Use $\sqrt{B} \times \sqrt{B}$ tiles as above. The final configuration, this time, has the histogram values in the top left $\sqrt{B} \times \sqrt{B}$ sub RMESH (this is necessary as $B > N$ and the histogram array cannot fit in a single column of the RMESH, one element per processor). The modifications to the case $N < B < N^2$ are straightforward. The complexity is $O(\sqrt{B}\log(N/\sqrt{B}))$. When $B \ge N^2$, we can use the algorithm of Figure 13 to compute H in $O(N)$ time. So, except when $B \ge N^2$, we can compute H on an $N \times N$ RMESH in less time than it takes to sort N^2 elements on such an RMESH. Notice that when $B \ge N^2$

$H[0]$	$H[4]$	$H[8]$	$H[12]$	$H[0]$	$H[4]$	$H[8]$	$H[12]$	$H[0]$	$H[4]$	$H[8]$	$H[12]$	$H[0]$	$H[4]$	$H[8]$	$H[12]$
$H[1]$	$H[5]$	$H[9]$	$H[13]$	$H[1]$	$H[5]$	$H[9]$	$H[13]$	$H[1]$	$H[5]$	$H[9]$	$H[13]$	$H[1]$	$H[5]$	$H[9]$	$H[13]$
$H[2]$	$H[6]$	$H[10]$	$H[14]$	$H[2]$	$H[6]$	$H[10]$	$H[14]$	$H[2]$	$H[6]$	$H[10]$	$H[14]$	$H[2]$	$H[6]$	$H[10]$	$H[14]$
$H[3]$	$H[7]$	$H[11]$	$H[15]$	$H[3]$	$H[7]$	$H[11]$	$H[15]$	$H[3]$	$H[7]$	$H[11]$	$H[15]$	$H[3]$	$H[7]$	$H[11]$	$H[15]$
$H[0]$	$H[4]$	$H[8]$	$H[12]$	$H[0]$	$H[4]$	$H[8]$	$H[12]$	$H[0]$	$H[4]$	$H[8]$	$H[12]$	$H[0]$	$H[4]$	$H[8]$	$H[12]$
$H[1]$	$H[5]$	$H[9]$	$H[13]$	$H[1]$	$H[5]$	$H[9]$	$H[13]$	$H[1]$	$H[5]$	$H[9]$	$H[13]$	$H[1]$	$H[5]$	$H[9]$	$H[13]$
$H[2]$	$H[6]$	$H[10]$	$H[14]$	$H[2]$	$H[6]$	$H[10]$	$H[14]$	$H[2]$	$H[6]$	$H[10]$	$H[14]$	$H[2]$	$H[6]$	$H[10]$	$H[14]$
$H[3]$	$H[7]$	$H[11]$	$H[15]$	$H[3]$	$H[7]$	$H[11]$	$H[15]$	$H[3]$	$H[7]$	$H[11]$	$H[15]$	$H[3]$	$H[7]$	$H[11]$	$H[15]$
$H[0]$	$H[4]$	$H[8]$	$H[12]$	$H[0]$	$H[4]$	$H[8]$	$H[12]$	$H[0]$	$H[4]$	$H[8]$	$H[12]$	$H[0]$	$H[4]$	$H[8]$	$H[12]$
$H[1]$	$H[5]$	$H[9]$	$H[13]$	$H[1]$	$H[5]$	$H[9]$	$H[13]$	$H[1]$	$H[5]$	$H[9]$	$H[13]$	$H[1]$	$H[5]$	$H[9]$	$H[13]$
$H[2]$	$H[6]$	$H[10]$	$H[14]$	$H[2]$	$H[6]$	$H[10]$	$H[14]$	$H[2]$	$H[6]$	$H[10]$	$H[14]$	$H[2]$	$H[6]$	$H[10]$	$H[14]$
$H[3]$	$H[7]$	$H[11]$	$H[15]$	$H[3]$	$H[7]$	$H[11]$	$H[15]$	$H[3]$	$H[7]$	$H[11]$	$H[15]$	$H[3]$	$H[7]$	$H[11]$	$H[15]$
$H[0]$	$H[4]$	$H[8]$	$H[12]$	$H[0]$	$H[4]$	$H[8]$	$H[12]$	$H[0]$	$H[4]$	$H[8]$	$H[12]$	$H[0]$	$H[4]$	$H[8]$	$H[12]$
$H[1]$	$H[5]$	$H[9]$	$H[13]$	$H[1]$	$H[5]$	$H[9]$	$H[13]$	$H[1]$	$H[5]$	$H[9]$	$H[13]$	$H[1]$	$H[5]$	$H[9]$	$H[13]$
$H[2]$	$H[6]$	$H[10]$	$H[14]$	$H[2]$	$H[6]$	$H[10]$	$H[14]$	$H[2]$	$H[6]$	$H[10]$	$H[14]$	$H[2]$	$H[6]$	$H[10]$	$H[14]$
$H[3]$	$H[7]$	$H[11]$	$H[15]$	$H[3]$	$H[7]$	$H[11]$	$H[15]$	$H[3]$	$H[7]$	$H[11]$	$H[15]$	$H[3]$	$H[7]$	$H[11]$	$H[15]$

Figure 17 Partition values for a 16×16 RMESH

only $O(1)$ memory per processor is needed.

4.1.2 $O(1)$ Memory Algorithm

We explicity consider only the case $B = N$. The algorithm differs from that for the $O(\sqrt{B})$ memory case only in the way the partial histogram values H are computed. This time, we sort the I values in each $\sqrt{N} \times \sqrt{N}$ sub RMESH, determine the head and tail of each sequence of equal I values, then determine the length of each such sub sequence using two concentrations as in

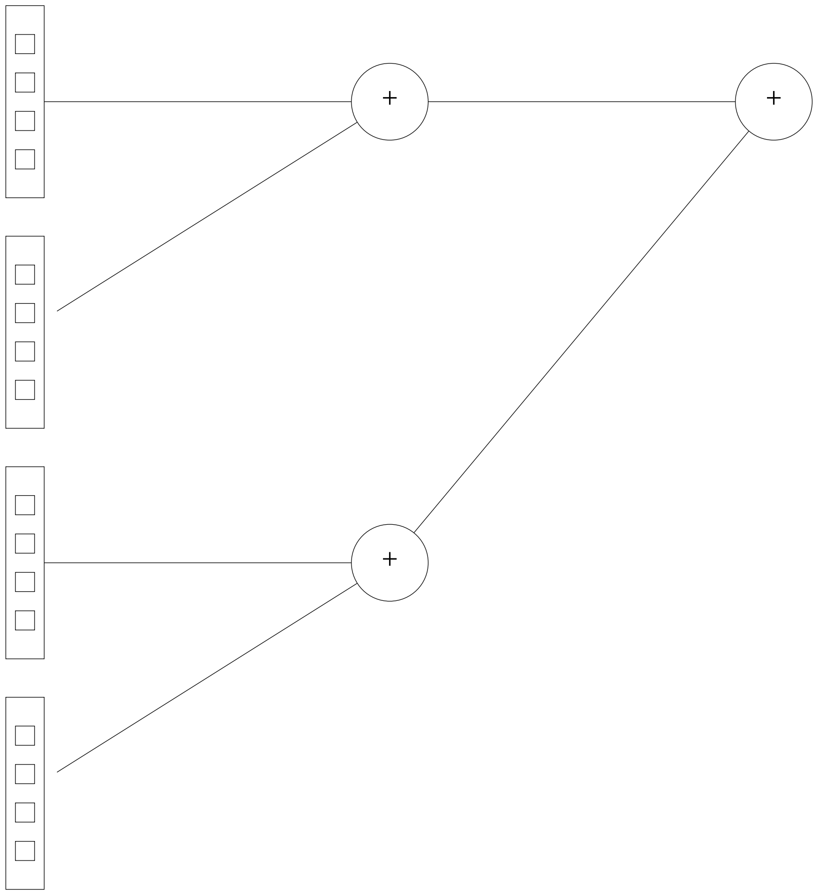


Figure 18 Adding blocks of $\sqrt{n} = 4$ processors in a column

Figure 13. All of this takes $O(\sqrt{N})$ time on a $\sqrt{N}\times\sqrt{N}$ RMESH. Next a random access write [NASS81] is used to route these H values to the appropriate processors so as to get the configuration of H values that results from Figure 16 and a consecutive sum. This also takes $O(\sqrt{N})$ time. A column sum and row sum, as for the case of the $O(\sqrt{B})$ memory algorithm, is needed to complete the computation. Since these summing operations take $O(\sqrt{N}\log N)$ time, the overall complexity becomes $O(\sqrt{N}\log N)$. The sort and random access write operations require many more broadcast steps than used in Figure 16 and in a consecutive sum. Hence the $O(1)$ memory algorithm is less efficient, in terms of time, than the $O(\sqrt{B})$ memory algorithm.

4.2 Histogram Modification

One of the operations commonly performed following the computation of the histogram of an image is histogram modification. In this the original gray values of the pixels are mapped to new gray values according to some function $f(OldGrayValue) \rightarrow NewGrayValue$, $0 \le OldGrayValue < B$ [LIM84]. This mapping is done in such a way that the new histogram approximates a desired distribution. Histogram modification is often done to improve the utilization of the available range of the gray scale and enhance the contrast of the image. For example, the histogram of an underexposed photograph is biased towards the darker gray levels.

Histogram modification is done in two steps:

1) Compute f

2) Update the gray values according to f

We consider two cases for the computation of f. In both we limit our discussion to the case $B = N$. Our algorithms are easily extended to other values of B. The two cases we consider are: histogram flattening and histogram modification by a prespecified distribution (e.g., normal distribution).

In histogram flattening (also known as histogram equalization) [PAVL82], the function f is obtained in the following way. Let $S[i] = \sum_{j=0}^{i} H[j]$, $0 \le i < B$, I.e., $S[0..N-1]$ gives the prefix sums of the histogram values. $f(i)$ is defined as :

$$f(i) = \lfloor S[i]/N \rfloor, \; 0 \le i < N$$

$$\text{or} f(i) = \lfloor (S[i] + S[i-1])/(2N) \rfloor, \; 0 \le i < N$$

Regardless of which definition is used, we can easily compute f on an $N \times N$ RMESH in $O(logN)$ time. Since the algorithm of Section 4.1 leaves H in column 0 of the RMESH, the computation of f involves a prefix sum of H values in column 0, followed by a possible shift by 1 of the prefix sum values (in case the second definition of f is used), and then some simple arithmetic. The prefix sum takes $O(logN)$ time and the shift $O(1)$ time. The function values f are left in column 0 of the RMESH. In the case of histogram modification by a prespecified distribution, f is computed by first obtaining the prefix sums $y_0, y_1, \cdots, y_{B-1}$ of the desired distribution. Let $s_0, s_1, ..., s_{B-1}$ be the prefix sums of the histogram H. For any i, $0 \le i < B$, $f(i)$ is defined to be an integer k such that $|y_k - s_i|$ is minimum over all k. We assume that ties are broken by picking the smallest k which minimizes $|y_k - s_i|$. For the computation of f we assume an initial condition in which the H values are stored, one value per processor, in column 0 of the $N \times N$ RMESH and the distribution values are stored, one value per processor, in row 0 of the RMESH. The algorithm to compute f is given in Figure 19. It is self explanatory and its complexity is $O(logN)$. The correctness of step 7 follows from the observation that since the y_i's form a montonic sequence, the d's form a bitonic sequence.

Now that we have seen how to compute f, we can consider the updating of the gray values as prescribed by f. The algorithm for this is given in Figure 20. In step 1 each processor accumulates $\sqrt{N}$ f values. Specifically, if PE (i,j) is the (a,b)'th processor in its $\sqrt{N}\times\sqrt{N}$ sub RMESH, then $U[k](i,j) = f(b\sqrt{N} + k)$, $0 \le i,j < N$. So, the U arrays in all processors in the same column of the $N\times N$ RMESH are the same. Processors in column j will contain, in their U array, the values $f(l)$, $(j \bmod \sqrt{N})\sqrt{N} \le l < (j \bmod \sqrt{N})\sqrt{N} + \sqrt{N}$. Specifically, we require

$$U[k](i,j) = f((j \bmod \sqrt{N})\sqrt{N} + k),\ 0 \le k < \sqrt{N},\ 0 \le i,j < N.$$

Notice that the range of U values in a processor corresponds to the range of A values computed in Figure 16. The actual updating of the gray values is done in step 2 of Figure 20. The process is somewhat similar to that used to obtain the A array in Figure 16. The I values are updated by considering the pixels in each $\sqrt{N}\times\sqrt{N}$ sub RMESH in column order. When the pixels in column k of the sub RMESHs are being considered, these pixels broadcast their gray values along row buses that are local to the sub RMESHs. The processors in each row of a sub RMESH contain the entire f function in their U arrays, $\sqrt{N}$ values per processor. When a processor receives a gray value, $V(i,j)$, it determines if this value is included in its range of f values. The f value range in PE (i,j) is from $(j \bmod \sqrt{N})\sqrt{N}$ through $(j \bmod \sqrt{N})\sqrt{N} + \sqrt{N}-1$. If the gray value is in its range, PE (i,j) determines the new gray value corresponding to this old gray value. The new gray value is then broadcast back to the PE that originated the old gray value and the old value is finally updated by the new one.

The complexity of the algorithm of Figure 20 is readily seen to be $O(\sqrt{N})$.

5 Conclusions

We have developed $O(\log N)$ RMESH algorithms to compute the area and perimeter of image components on an $N\times N$ RMESH. For the histogram problem we have developed an $N\times N$ RMESH algorithm that is faster than sorting when the number, B, of gray scale values is less than N^2. The time complexity of this algorithm is $O(\min\{\sqrt{B}\log(N/\sqrt{B}), N\})$ and it uses $O(\sqrt{B})$ memory per processor. Another $N\times N$ RMESH algorithm using $O(1)$ memory per processor was also developed. This has complexity $O(\sqrt{N}\log N)$ where $N = B$. Finally, we showed how

Step 1 Prefix sum the distribution values in row 0 of the RMESH.

The prefix sums are stored in $y(0,j)$, $0 \le j < N$.

Step 2 Prefix sum the histogram values in column 0 of the RMESH.

These sums are stored in $s(i, 0)$, $0 \le i < N$.

Step 3 Set up column buses and broadcast the $y(0,j)$ values, $0 \le j < N$.

$y(i,j)$:= content(bus), $0 \le i,j < N$; $\{y(i,j) = y_j\}$

Step 4 Set up row buses and broadcast the $s(i, 0)$ values, $0 \le i < N$.

$s(i,j)$:= content(bus), $0 \le i,j < N$;$\{s(i,j) = s_i\}$

Step 5 $d(i,j) := | y(i,j) - s(i,j)) |$, $0 \le i,j < N$.

Step 6 Shift the d values left on rows by 1, the shifted values are in the e variables, I.e., $e(i,j) = d(i,j+1)$, $0 \le i < N$, $0 \le j < N-1$

Step 7 **if** $d(i,j) < e(i,j)$ **then** PE (i,j) disconnects its E switch and then broadcasts j; $0 \le i < N$, $0 \le j < N-1$;

PE $(i,N-1)$ broadcasts $N-1$, $0 \le i < N$;

Step 8 $f(i, 0)$:= content(bus);

Figure 19 Computing f according to a given distribution

histogram modification could be done on an $N \times N$ RMESH in $O(\sqrt{N})$ time.

6 References

[BEST89] T. Bestul and L. S. Davis, "On computing complete histograms of images in log(n) steps using hypercubes", IEEE Transactions on Pattern Analysis and Machine Intelligence, vol 11, no 2, pp 212-213.

[DUDA73] R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis*, John Wiley and Sons, 1973.

[GRIN84] J. Grinberg, G. R. Nudd and R. D. Etchells "A cellular VLSI architecture" IEEE Computer, Jan. vol. 17, no. 1, 1984, pp 69-81.

Step 1 { Obtain the mapping array U in each processor }

set up row buses;

PE (i, 0) broadcasts $f(i, 0)$, $0 \le i < N$;

$g(i,j)$:= content(bus), $0 \le i,j < N$;

set up column buses;

for k := 0 to $\sqrt{N}-1$ **do**

begin

PE $((j \ mod \ \sqrt{N})\sqrt{N} + k, j)$ broadcasts its g value, $0 \le j < N$;

$U[k](i,j)$:= content (bus), $0 \le i,j < N$;

end;

Step 2 { Update gray values }

set up row buses local to each $\sqrt{N} \times \sqrt{N}$ sub RMESH;

for k := 0 **to** $\sqrt{N}-1$ **do**

begin

PE (i,j) broadcasts its I value, $0 \le i,j < N, j \ mod \sqrt{N} = k$;

$V(i,j)$:= content(bus), $0 \le i,j < N$;

if $(j \ mod \ \sqrt{N})\sqrt{N} \le V(i,j) < (j \ mod \ \sqrt{N})\sqrt{N} + \sqrt{N}$

then broadcast $U[V(i,j) \ mod \ \sqrt{N}](i,j)$;

$I[i,j]$:= content(bus); $0 \le i,j < N, \ j \ mod \ \sqrt{N} = k$;

end;

Fgiure 20 Updating gray values

[HORO78] E. Horowitz and S. Sahni, *Fundamentals of Computer Algorithms*, Computer Science Press, Inc., 1978.

[LIM84] J. S. Lim, "Image enhancement", in *Digital Image Processing Techniques*, Ed. M. P. Ekstrom, Academic Press, 1984, pp 1-51.

[MILL88a] R. Miller, V. K. Prasanna Kumar, D. Resis and Q. Stout, "Data movement operations and applications on reconfigurable VLSI arrays", Proceedings of the 1988 International Conference on Parallel Processing, The Pennsylvania State University Press, pp 205-208.

[MILL88b] R. Miller, V. K. Prasanna Kumar, D. Resis and Q. Stout, "Meshes with reconfigurable buses", Proceedings 5th MIT Conference On Advanced Research IN VLSI, 1988, pp 163-178.

[MILL88c] R. Miller, V. K. Prasanna Kumar, D. Resis and Q. Stout, "Image computations on reconfigurable VLSI arrays", Proceedings IEEE Conference On Computer Vision And Pattern Recognition, 1988, pp 925-930.

[NASS79] D. Nassimi and S. Sahni, "Bitonic sort on a mesh connected parallel computer", IEEE Transactions on Computers, vol C-27, no. 1, Jan. 1979, pp 2-7.

[PAVL82] T. Pavlidis "Algorithms for graphics and image processing", Computer Science Press, Maryland, 1982.

[SIEG81] H. J. Siegel, L. Siegel, F. C. Kemmerer, P. T. Muller, H. E. Smalley, and S. D. Smith "PASM: A partitionable SIMD/MIMD system for image processing and pattern recognition", IEEE Transactions on computers, vol. C-30, no. 12, Dec. 1981, pp 934-947.

[TANI84] S. L. Tanimoto, "Sorting, histogramming and other statistical operations on a pyramid machine", in *Multiresolution image processing and analysis*, Ed. A. Rosenfeld, Springer-Verlag, New York, 1984, pp 136-145.

[YASR83] M. Yasrebi, S. Deshpande and J. C. Browne, "A comparison of circuit switching and packet switching data transfer using two simple image processing algorithms", Proceedings 1983 International Conference on Parallel Processing, IEEE Computer Society Press, pp 25-28.