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

**Abstract**

Parallel reconfigurable mesh computer algorithms are developed to obtain the area and perimeter of image components. For an $N \times N$ image, our algorithms take $O(logN)$ time on an $N \times N$ RMESH.

**Keywords and Phrases**

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

## 1 Introduction

Miller, Prasanna Kumar, Reisis and Stout [4, 5, 6] 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. This is described in our companion paper [3] which is included in these proceedings. In this paper, we develop RMESH algorithms to compute the area and perimeter of the components in an image. In several applications [1] 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 border).

## 2 Area And Perimeter Of Connected Components

Initially, each pixel of the binary image matrix $I[0..N-1,0..N-1]$ 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 for area determination. The pixels are first sorted by the field *comp*. Next the first and last pixel of each sequence with the same *comp* value is identified. The distance between these can be obtained by performing a data concentration [7] 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 $\Theta(N)$ time to sort $N^2$ elements.

### 2.1 Area

#### 2.1.1 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 [2]. 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. In the other, two vertically adjacent $2^i \times 2^{i+1}$ blocks are combined. 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 horizontally adjacent blocks and pairs of vertically adjacent blocks. With each pixel $[i,j]$, we associate two additional fields: *update* and *area*. *update* is a Boolean field and *area* is an integer field which will eventually be the number of 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. Following each combination, the following is true:

> 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. 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:

$$I[i,j].area = \text{number of pixels in the block with } comp \text{ value equal to } I[i,j].comp$$

If $[i,j]$ is a boundary pixel of block $A$ ($B$) and $I[i,j].comp > 0$, then its *area* value changes if 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 $I[u,v].comp = I[i,j].comp \ne 0$. Then the updated *area* value for pixel $[i,j]$ 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

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

combination has the form $((Y(A), Z(B)),$ $Y, Z \in \{top, bottom, left, right\}$. The code of Figures 1 and 2 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 $1 \times 1$ and ending with a single block of size $N \times N$, the complexity of the procedure to compute area for boundary pixels is $O(logN)$.

---

$I[i,j].update :=$ 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 1** Combine blocks *A* and *B*

---

---

**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 2** Combining two boundary lines

---

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 $N^2$ $1{\times}1$ blocks.

---

**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 3** RMESH version of *CombineLines*

---

### 2.1.2 RMESH Algorithm

The RMESH algorithm works like the CRCW PRAM algorithm. We need to provide only the details for the code of Figure 2 (i.e., procedure CombineLines). Figure 3 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.

## 2.2 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.

## 3 References


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

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

[3]J. Jenq and S. Sahni, "Reconfigurable mesh algorithms for the Hough transform", *Proc. International Conf. on Parl. Processing*, 1991.

[4]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.

[5]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.

[6]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.

[7]D. Nassimi and S. Sahni, "Data broadcasting in SIMD computers", IEEE Transactions on Computers, vol C-30, no. 2, Feb. 1981, pp 101-107.