Parallel Processing for Artificial Intelligence 1
L.N. Kanal, V. Kumar, H. Kitano and C.B. Suttner (Editors)

Chapter 4

Image Processing On Reconfigurable Meshes With Buses*

Jing-Fu Jenq
Tennessee State University

and

Sartaj Sahni
University of Florida

Abstract

In this chapter, we describe different reconfigurable mesh with buses architectures and show how several image processing problems can be solved efficiently on the weakest of these. The specific problems considered are: area and perimeter of components, shrinking and expanding, clustering, and template matching. In many cases, the resulting algorithms are faster than those for other parallel computer architectures.

1. Introduction

Recently, several researchers have proposed a modification to the well studied mesh architecture in which the interprocessor links are replaced by a reconfigurable bus. The resulting parallel computer architecture is called a reconfigurable mesh with bus (RMB). In all two dimensional $N \times N$ RMB computers, the N^2 processors are located at the N^2 grid points of an $N \times N$ grid (just as in a traditional mesh computer). However, the traditional linkage between mesh adjacent processors is absent. Instead, interprocessor communication takes place via a reconfigurable bus. The RMB family of architectures includes the RMESH, PARBUS, polymorphic torus, and the mesh reconfigurable network (MRN). The architectures have become popular because they are ralatively easy to program and because many problems can be solved very efficiently on them. In fact, it is possible to solve some problems faster on an RMB computer than is theoretically possible on a PRAM computer (See for example: [10, 34, 35, 39]).

*This research was supported, in part, by the National Science Foundation under grant MIP-9103379.

In the RMESH [53, 26, 27, 28] version of an RMB (Fig. 1), the bus is comprised of $2N(N-1)$ segments that correspond to the $2N(N-1)$ interprocessor links in a traditional mesh. However, each of these segments has a switch on it. The switch on a bus segment may be set in the open or closed position by either one of the two processors at the ends of the segment. With the switch on each segment closed, all N^2 processors are attached to the same bus. As a result, if any one processor writes data to this bus, all remaining processors can read this data from the bus in the next cycle. I.e., it is possible to broadcast data in O(1) time.

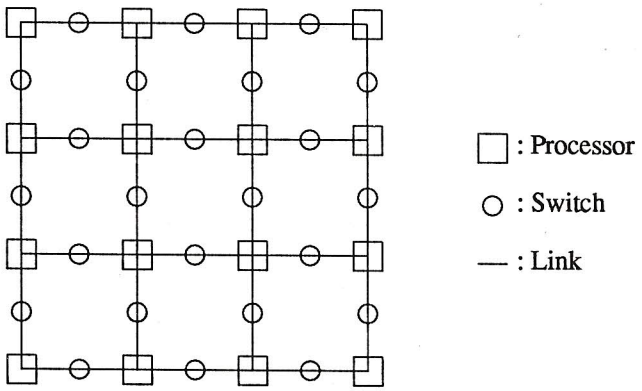
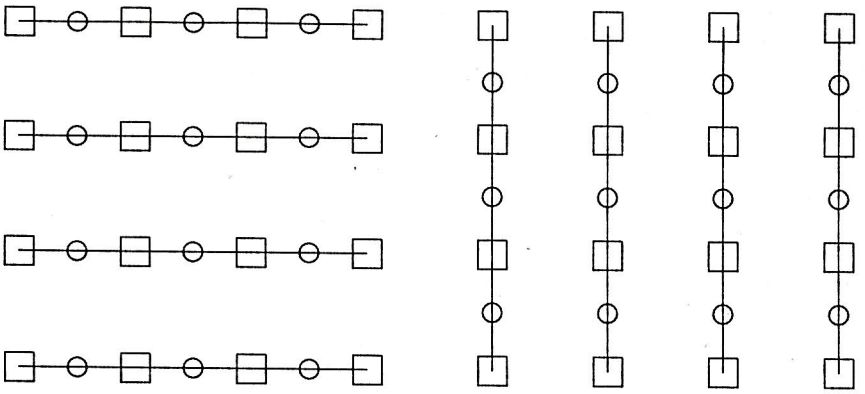


Fig. 1 A 4×4 RMESH

By opening the switches on all vertical bus segments and closing them on all horizontal ones, we form N independent buses with each one spanning the processors in a row of the mesh (Fig. 2(a)). Column buses that span all processors in the same column may be formed by closing all switches on column segments and opening all on row segments (Fig. 2(b)). As in other work dealing with the RMESH model, we assume that the the time to broadcast data on a bus or subbus is O(1). In the exclusive write model, only one processor can write data to a given (sub)bus at any time. In the concurrent write model several processors may simultaneously write to the same (sub)bus. Rules are provided to determine which of the several writers actually succeeds (e.g., arbitrary, maximum, exclusive or, etc.).

The PARBUS of [51, 52] is also a member of the RMB family. An $N \times N$ PARBUS (Fig. 3) is an $N \times N$ mesh in which the interprocessor links are bus segments and each processor has the ability to connect together arbitrary subsets of the four bus segments that connect to it. Bus segments that get so connected behave like a single bus. The bus segment interconnections at a proccessor are done by an internal four port switch. If the up to four bus segments at a processor are labeled N (North), E (East), W (West), and S (South), then this switch is able to realize any set, A = $\{A_1, A_2\}$, of connections where $A_i \subseteq$ {N,E,W,S}, $1 \leq i \leq 2$ and the A_i's are disjoint. For example A = {{N,S}, {E,W}}



(a) Row buses (b) Column buses

Fig. 2 Subbuses in an RMESH

results in connecting the North and South segments together and the East and West segments together. If this is done in each processor, then we get, simultaneously, disjoint row and column buses. If A = {{N,S},ϕ}, then only column buses are formed. Similarly, when A = {{E,W}, ϕ} only row buses are formed. If A = {{N,S,E,W},ϕ}, then all four bus segments are connected. PARBUS algorithms for a variety of applications can be found in [29, 52, 23, 10, 11, 12, 13, 36, 37, 38, 49]. Observe that in an RMESH the realizable connections are of the form A = $\{A_1\}, A_1 \subseteq$ {N,E,W,S}.

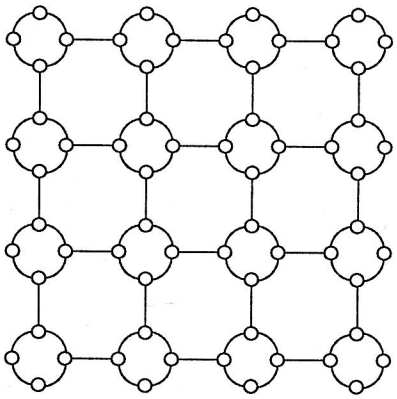


Fig. 3 A 4 × 4 PARBUS

The polymorphic torus and the mesh reconfigurable network (MRN) are

two other members of the RMB family. The polymorphic torus architecture [21, 22] is identical to the PARBUS except that the rows and columns of the underlying mesh wrap around and it possible able to connect together arbitrary subsets of the bus segments that connect to the processor. Specifically, in each row, there is an additional bus segment that connects the rightmost port in the row to the leftmost port in the row and in each column there is a bus segment that connects the lowest port in the column to the topmost one. In an MRN [2], the processor and bus segment arrangement is exactly as for the PARBUS. However the switches internal to processors are able to obtain only the 10 bus configurations given in Fig. 4. Thus an MRN is a restricted PARBUS.

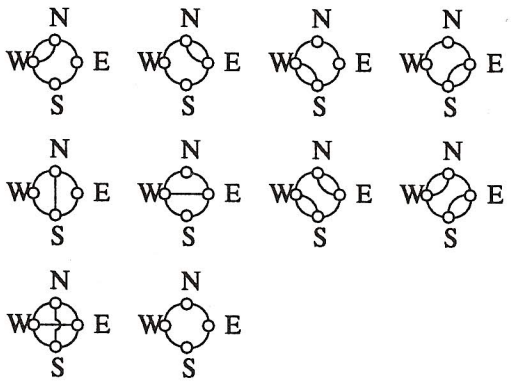


Fig. 4 Allowable switch configurations in an MRN

While we have defined the above reconfigurable bus architectures as square two dimensional meshes, it is easy to see how these may be extended to obtain non square architectures and architectures with more dimensions than two. In this chapter, we shall confine ourselves to image processing algorithms that run on an RMESH. While these same algorithms can be run on a PARBUS, polymorphic torus, and an MRN with no loss in run time, one can often obtain faster algorithms for these other architectures. The reader is referred to the cited references for work on these other architectures. MacKenzie [MACK93] has shown that there are problems that can be solved faster on the other models than on the RMESH. Furthermore, Jang et al. [13] have shown how many (though not all) of the connections supported by the remaining models can be simulated by an RMESH with no loss in run time. However, the simulating RMESH needs 16 processors for each processor in the simulated model and the required I/O configuration must be modified so that data is at just one processor in each 4×4 group of RMESH processors.

In this chapter, we shall study RMESH algorithms for the following image processing applications: area and perimeter of components, shrinking and expanding, clustering, and template matching. RMESH algorithms for

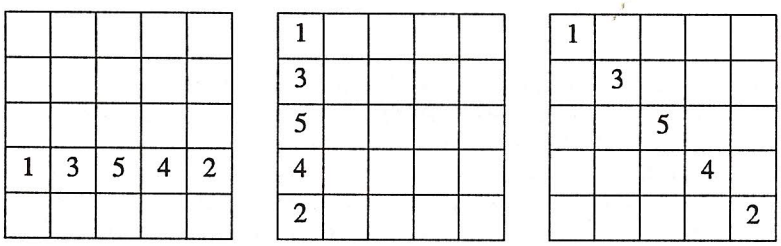
histogramming, histogram modification, and the Hough transform can be found in [17] and [15]. It should be noted that Jang et al. [11] have developed a histogramming algorithm for a PARBUS. They state that the algorithm can be simulated on an RMESH. However, this simulation requires a 16 fold increase in the number of processors and requires that the input be properly configured. To obtain the initial configuration from one in which the $N \times N$ image resides in an $N \times N$ block of the $4N \times 4N$ RMESH requires O(N) time as the bandwidth available for the rearrangement is only O(N). This rearrangement time exceeds the total run time of our algorithms which require only an $N \times N$ RMESH.

RMESH algorithms for some other image processing problems can be found in [26, 27, 28]. [34] develops constant time RMESH algorithms for some problems from computational geometry.

2. Data Manipulation Operations

2.1. Data Diagonalization

In this, a specific row or column of elements is moved to the diagonal positions of the window which contains that row or column. This is illustrated in Fig. 5. This can be accomplished in O(1) time by broadcasting the row (column) along column (row) buses and having the diagonal processors read the data from their bus.



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

Fig. 5 Diagonalize 4th row or 1st column elements of a 5×5 window

2.2. 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 $A(i,j) = A(i \bmod w, j \bmod w)$ ($A(i,j)$ denotes register A of the PE with ID (i,j)). This can be done in $O(w)$ independent of the size of the RMESH [18].

2.3. Prefix Sum

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

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

An $O(\log N)$ algorithm for this is given in [26].

2.4. 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 :

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

This can be done in $O(\log N)$ time by first performing a prefix sum [26] 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.

2.5. Ranking

Consider the linear ordering of the N^2 PEs defined by row major order. I.e., PE $[i,j]$ is in position $iN+j$ of this ordering. Assume that each PE has a Boolean variable *selected*. If $selected(i,j)$ is true then $rank(i,j)$ is the number of PEs with $selected(i,j)$ true that precede it in the defined linear ordering. If $selected(i,j)$ is false, then $rank(i,j)$ is undefined. N^2 elements can be ranked using N^2 processors in O(log$N$) time using the prefix sum operation. However, N elements on a single column or row can be ranked in O(1) time on an $N \times N$ RMESH [18].

2.6. 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

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

A circular shift variant of the above shift requires

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

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 $a = i\ mod\ W$'th and $b = q\ mod\ W$'th PEs in the same $W\times 1$ column window and $b = (a-s)\ mod\ W$.

All variants of shifts described above can be done in $O(s)$ time [18].

2.7. Data Accumulation

In this operation PE (i,j) initially has a value $I(i,j)$, $0 \le i,j < N$. Each PE is required to accumulate M I values in its array A as specified below:

$$A[q](i,j) = I(i,\ (j+q)\ mod\ N)$$

This can be done using $2M-1$ broadcasts [18].

2.8. 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 $j\ mod\ M$'th processor in its block. Each processor (i,j) of the RMESH has an array $X[0..M-1](i,j)$ of values. If $j\ mod\ M = q$, then PE (i,j) is to compute $S(i,j)$ such that

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

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 can be done using $3M-3$ broadcasts [18].

2.9. Adjacent Sum

There are two forms of this operation: row adjacent sum and column adjacent sum. In each, PE (i,j) begins with an array $X[0..M-1](i,j)$ of values. In a row adjacent sum, PE (i,j) is to compute

$$S(i,j) = \sum_{q=0}^{M-1} X[q](i,\ (j+q)\ mod\ N),\ \ 0 \le i,j < N$$

While in a column adjacent sum it is to compute

$$S(i,j) = \sum_{q=0}^{M-1} X[q]((i+q) \bmod N, j), \quad 0 \le i,j < N$$

Both forms of adjacent sum can be done with $3(M-1)$ broadcasts [18].

2.10. 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 [31]. When N elements are to be sorted on an $N \times N$ RMESH the sort can be accomplished in O(1) time [35]. The initial and final configuration has the data in row 0 of the $N \times N$ RMESH.

2.11. RAR and RAW

The random access read (RAR) and random access write (RAW) operations are defined in [32]. 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. [26] 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, $k \le N^2$. If the number of source and destination processors in each $k \times k$ block of PEs is $O(k)$, $1 \le k \le N$ then their algorithm takes $O(\log N)$ time.

When the source and destination processors are all on a single row of an $N \times N$ RMESH, then RARs and RAWs can be done in $O(1)$ time [18].

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.*

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 $I[i,j].comp > 0$. On an $N \times N$ RMESH, the area and perimeter can be determined in $O(\log N)$ time [16]. These operations can be performed on O(1) time

on a PARBUS with $O(p*n^{2+\varepsilon})$ and $O(\max\{n,p\}*n^{2+\varepsilon})$ processors where p is the number of image components and ε is any fixed constant greater than zero [23]. We develop the RMESH algorithm of [16] here.

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 [8]. 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: horizontal and vertical. When two horizontally adjacent blocks of size $2^i \times 2^i$ each are combined, we get a single block of size $2^i \times 2^{i+1}$. 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.

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 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. Following each combination, the following is true:

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. 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), and *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 Figs. 6 and 7 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×N, the complexity of the procedure to compute area for boundary pixels is O (log$N$).

```
I[i,j].update := false, 0 ≤ i,j < N
for sideA ∈ {top, bottom, left, right} do
  for sideB ∈ {top, bottom, left, right } do
    CombineLines(sideA,sideB);
```

Fig. 6 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 ≤ c < |sideA| of sideA of A
and the d'th pixel, 0 ≤ 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;
```

Fig. 7 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 1×1 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 is similar to that described earlier for block combination and we omit the details.

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 Fig. 7 (i.e., procedure CombineLines). Fig. 8 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 $O(\log N)$ time.

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 $O(\log N)$ algorithm of the preceding section can be used.

4. Shrinking And Expanding

Let $I[0..N-1,0..N-1]$ be an $N \times N$ image and let $B_{2q+1}[i,j]$ represent the block of pixels:

$$\{[u,v] \mid 0 \le u,v < N, \max\{\,|u-i|, |v-j|\,\} \le q\}$$

Rosenfeld [48] has shown that the *q-step expansion*, E^q, and the *q-step shrinking*, S^q, of I are given by the equations:

$$E^q[i,j] = \max_{[u,v] \in B_{2q+1}[i,j]} \{I[u,v]\},\ 0 \le i,j < N$$

$$S^q[i,j] = \min_{[u,v] \in B_{2q+1}[i,j]} \{I[u,v]\},\ 0 \le i,j < N$$

Rosenfeld [48] presents an algorithm for the pyramid computer which computes S^{2^k-1} and E^{2^k-1} at coarsely resampled points in $O(k)$ time. The complexity is valid for both binary and gray scale images. For unresampled binary

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*;

Fig. 8 RMESH version of *CombineLines*

image expanding and shrinking in one dimension he developed an $O(k^2)$ algorithm to compute S^{2^k-1} and E^{2^k-1}. The generalization to two dimensions results in a complexity of $O(2^k)$. Ranka and Sahni [44] show how S^{2^k-1} and E^{2^k-1} may be computed in $O(k)$ time on an N^2 PE SIMD hypercube. Their algorithms apply to both binary and gray scale images. E^q and S^q are easily computed in $O(q)$ time on an $N\times N$ mesh connected computer. In this section we develop an algorithm to compute E^q in $O(1)$ time on an $N\times N$ RMESH. Our algorithm is for the case of a binary image. S^q, for binary images, can be similarly computed in $O(1)$ time.

Since shrinking and expanding are computationally equivalent, we consider only expansion explicitly. From the equation for E^q, it follows that

$$E^q[i,j] = \max_{[u,j]\in B_{2^q+1}[i,j]} \{R^q[u,j]\} \ ,0 \le i,j < N$$

where

$$R^q[u,j] = \max_{[u,v]\in B_{2q+1}[i,j]} \{I[u,v]\}\ ,0 \le u,j < N$$

$$= \max\ \{\ I[u,v]\ |\ \ |j-v| \le q\ \},\ 0 \le u,j < N$$

The computation of R^q may be decomposed into subcomputations as below:

$$left^q[u,j] = \max\{I[u,v] \mid 0 \le j-v \le q\}$$

$$right^q[u,j] = \max\{I[u,v] \mid 0 \le v-j \le q\}$$

$$R^q[u,j] = \max\{left^q[u,j],\ right^q[u,j]\}$$

Similarly we may decompose the computation of E^q from R^q as below:

$$top^q[i,j] = \max\{R^q[u,j] \mid 0 \le i-u \le q\}$$

$$bottom^q[i,j] = \max\{R^q[u,j] \mid 0 \le u-i \le q\}$$

$$E^q[i,j] = \max\{top^q[i,j],\ bottom^q[i,j]\}$$

{Compute $R^q[i,j]$ in variable R of PE $[i,j]$}
{Assume that $I(i,j) = I[i,j]$ initially}

Step 1 {Compute $left^q[i,j]$ in variable *left* of *PE* (i,j) }
{Find nearest 1 on the left }
PE (i,j) disconnects its N and S switches
if $I(i,j) = 1$ **then** *PE* (i,j) disconnects its W switch and broadcasts j on its bus

Step 2 PE (i,j) reads its bus and puts the value read in its T variable
if $j - T(i,j) \le q$ **then** $left(i,j) = 1$
else $left(i,j) = 0$

Step 3 {Compute $right^q[i,j]$ by finding nearest 1 on right }
PE (i,j) connects its E and W switches.
if $I(i,j) = 1$ **then** *PE* (i,j) disconnects its E switch and broadcasts j on its bus
{N and S switches are disconnected from Step 1}

Step 4 *PE* (i,j) reads its bus and puts the value read in its T variable
if $T(i,j) - j \le q$ **then** $right(i,j) = 1$
else $right(i,j) = 0$

Step 5 {Compute R}
$R(i,j) :=$ $left(i,j)$ **or** $right(i,j)$

Fig. 9 Computing R for a binary image

The steps involved in computing R^q for a binary image I are given in Fig. 9. The complexity is readily seen to be $O(1)$. E^q is similary computed from R^q in $O(1)$ time. The algorithm of Fig. 9 assumes that all switches are initially connected and that if a processor reads a bus and finds no value, the value ∞ is

used.

5. Clustering

The input to the clustering problem is an $N \times M$ *feature matrix* $F[0..N-1, 0..M-1]$. Row i of F defines the *feature vector* for pattern i. Thus there are N patterns represented in F. Each column of F corresponds to a pattern attribute. Thus, M is the number of pattern attributes. The objective of clustering is to partition the N patterns into K sets $S_0, S_1, \cdots, S_{K-1}$. Each S_i is called a *cluster*. Different methods to cluster have been proposed in [1], [4], [7], [6], [47], and [50]. Here, we consider the popular squared error clustering technique. In this we begin with an initial (possibly random) partitioning (i.e. clusters) of the patterns and iteratively improve the clusters as described below.

The *center* of cluster S_i is a $1 \times M$ vector which gives the average of the attribute values for the members of this cluster. The centers of the K clusters may be represented by a $K \times M$ matrix $C[0..K-1, 0..M-1]$ where $C[i,*]$ is the center of the i'th cluster S_i and

$$C[i,j] = \frac{1}{|S_i|} \sum_{q \in S_i} F[q,j], \quad 0 \le i < K, \; 0 \le j < M$$

The *squared distance*, $d2[i,k]$, between pattern i and cluster k is defined to be the quantity

$$d2[i,k] = \sum_{q=0}^{M-1} (F[i,q] - C[k,q])^2$$

One pass of the iterative cluster improvement process is given in Fig. 10.

Step 1 [Cluster reassignment]
Newcluster $[i] := j$ such that $d2[i,j] = \min_{0 \le q < k} \{d2[i,q]\}, \; 0 \le i < N$
[In case of a tie pick the least j]

Step 2 [Termination and update]
if *NewCluster* $[i]$ = *OldCluster* $[i], \; 0 \le i < N$
then terminate
else *OldCluster* $[i]$ = *NewCluster* $[i], \; 0 \le i < N$

Step 3 [Update cluster centers]
Compute $C[*,*]$ based on the new clusters

Fig. 10 One pass of the iterative cluster improvement algorithm

The serial complexity of one pass of the iterative cluster improvement algorithm is readily seen to be $O(NMK)$. Parallel algorithms for this have been developed by several researchers. Hwang and Kim [9] have developed an algorithm for a multiprocessor with orthogonally shared memory; Ni and Jain [33]

have proposed a systolic array; Li and Fang [20] have developed an $O(K\log NM)$ algorithm for an SIMD hypercube with NM processors; and Ranka and Sahni [41] have developed an $O(K + \log NMK)$ algorithm for an SIMD hypercube with NM processors as well as an $O(\log NMK)$ algorithm for an SIMD hypercube with NMK processors.

In [14], we have developed a clustering algorithm for RMESH's with N processors. The time complexity of this algorithm is $O(MK + K\log N)$. In the remainder of this section we consider RMESH's with NM and NMK processor. The algorithms for these cases have complexity $O(K\log MN)$, and $O(M + \log NMK)$, respectively.

5.1. *NM* processors

The NM processor RMESH is assumed to be configured as an $N\times M$ array of processors. Initially, $F(i,j) = F[i,j]$, $0 \le i < N$, $0 \le j < M$ and $C(i,j) = C[i,j]$, $0 \le i < K$, $0 \le j < M$. The algorithm to obtain the new cluster assignments is given in Fig. 11. Summing the E values takes $O(\log M)$ time. The remaining steps each take $O(1)$ time. The overall complexity is therefore $O(K\log M)$.

$D2(i, 0) := \infty$, $0 \le i < N$
for $i := 0$ **to** $K-1$ **do**
begin
  Set up column buses;
  PE (i,j) broadcasts $C(i,j)$ on its column bus, $0 \le j < M$;
  PE (a,b) reads its column and saves the value read in $D(a,b)$,
  $0 \le a < N$, $0 \le b < M$;
  $E(a,b) := (F(a,b) - D(a,b))^2$;
  Set up row buses;
  Sum the E values in row a and save in $S(a, 0)$, $0 \le a < N$;
  if $S(a, 0) < D2(a, 0)$ **then**
    [*NewCluster* $(a, 0) := i$; $D2(a, 0) := S(a, 0)$];
end;

Fig. 11 *NM* processors algorithm for cluster reassignment

Step 2 of Fig. 10 (i.e., terminate and update) is easily done in $O(1)$ time. The cluster centers may be updated in $O(K\log N)$ time using the algorithm of Fig. 12. The overall complexity of one pass of the iterative cluster improvement algorithm is therefore $O(K\log MN)$ on an NM processor RMESH.

5.2. *NMK* processors

The RMESH is configured as an $N\times MK$ array of processors with $F(i,j) = F[i,j]$, $0 \le i < N$, $0 \le j < M$ and $C(0,j) = C[j \text{ div } M, j \text{ mod } M]$, $0 \le j < KM$ initially. Our RMESH algorithm for cluster reassignment (Fig. 13)

for $i := 0$ **to** $K-1$ **do**
{Compute center of cluster i }
begin
Set up row buses;
if *NewCluster* $(a, 0) = i$ **then** PE $(a, 0)$ broadcasts a 1 on its row bus
else PE $(a, 0)$ broadcasts a 0, $0 \le a < N$;
PE (a,b) sets $A(a,b) = F(a,b)$ if it reads a 1 on its bus,
otherwise it sets $A(a,b) = 0$;
Sum the A values on each column and save the result in the C variable
of the processors in row i;
end;

Fig. 12 Computing cluster centers with NM processors

begins by making K copies of the feature matrix. This can be done using M broadcasts on row buses. The remainder of the alogrithm takes $O(\log MK)$ time. Hence the complexity of the algorithm of Fig. 13 is $O(M + \log MK)$. Step 2 of Fig. 10 is easily done in $O(1)$ time.

Step 1 Create K copies of F so that $F(i,j) = F[i, j \bmod M]$,
$0 \le j < KM$, $0 \le i < N$.
Step 2 Set up column buses and broadcast $C(0,j)$ to $C(*,j)$, $0 \le j < KM$.
Step 3 $A(i,j) = (F(i,j) - C(i,j))^2$, $0 \le i < N$, $0 \le j < MK$
Step 4 Sum A in each $1 \times M$ block of PEs. Store the result in the
B variable of the first processor in each block.
Step 5 In each row, a, of the RMESH, compute q such that
$B(a,q) = \min\{B(a,j) \mid j \bmod M = 0\}$;
store this q in *NewCluster* $(a, 0)$;

Fig. 13 New cluster determination with NMK processors

The new cluster centers can be computed in $O(\log N)$ time by summing the feature values in each cluster and dividing by the number of patterns in the cluster. Column j of the RMESH is used to compute $C[j \operatorname{div} M, j \bmod M]$. The complexity of one pass of the iterative improvement algorithm is therefore $O(M + \log NMK)$ when NMK processors are available.

6. Template Matching

In this section we develop RMESH algorithms for the image template matching problem. The inputs are an $N \times N$ image matrix $I[0..N-1, 0..N-1]$ and an $M \times M$ template matrix $T[0..M-1, 0..M-1]$. The ouput is the two dimensional convolution, C2D, of I and T which is defined as:

$$C2D[i,j] = \sum_{u=0}^{M-1} \sum_{v=0}^{M-1} I[(i+u) \bmod N, (j+v) \bmod N] * T[u,v], \quad 0 \le i,j < N$$

The serial complexity of computing C2D is $O(N^2M^2)$. Parallel algorithms for a variety of multiprocessor architectures have been developed. Chang et al. [3] have developed an $O(M^2 + \log N)$ algorithm for a pyramid computer with N^2 processors at its base. Ranka and Sahni [41] have developed an $O(M^2)$ algorithm for an N^2 processor mesh connected computer. Maresca and Li [25] and Lee and Agarwal [19] also develop mesh algorithms. Hypercube algorithms have been developed by Fang, Li and Ni [5], Prassanna Kumar and Krishnan [40], and Ranka and Sahni [42, 43]. The SIMD hypercube algorithm of [42] uses N^2 processors and has complexity $O(M^2 + \log N)$ and the MIMD hypercube algorithm of [43] uses N^2 processors and has complexity $O(M^2)$. Ranka and Sahni [43] also report on experimental work with image template matching algorithms for the NCUBE hypercube computer.

The RMESH algorithms we develop in this section have the following characteristics:

Algorithm 1: N^2 processors, $O(M)$ memory per processor, $O(M^2)$ complexity.

Algorithm 2: N^2 processors, $O(1)$ memory per processor, $O(M^2)$ complexity.

Algorithm 3: N^2M^2 processors, $O(1)$ memory per processor, $O(M)$ complexity.

While one can obtain $O(M^2)$ RMESH template matching algorithms that use N^2 processors by simulating the known mesh connected computer algorithm of this complexity, the algorithms we propose are considerably simpler.

6.1. N^2 Processors, $O(M)$ Memory

The N^2 processors are configured as an $N \times N$ array. Initially, $I(i,j) = I[i,j]$, $0 \le i,j < N$ and $T(i,j) = T[i,j]$, $0 \le i,j < M$. The desired final configuration is $C2D(i,j) = C2D[i,j]$, $0 \le i,j < N$. This is accomplished in two steps. First, each PE (i,j) computes an array $C[q](i,j)$, $0 \le q < M$ of values such that:

$$C[q](i,j) = \sum_{v=0}^{M} I[i, (j+v) \bmod N] * T[q,v]$$

Next, $C2D(i,j)$ is computed using the equation:

$$C2D(i,j) = \sum_{q=0}^{M-1} C[q]((i+q) \bmod N, j)$$

This is done by simply using the column adjacent sum operation of Section 2.9. The details are given in Fig. 14. The complexity is $O(M^2)$.

6.2. N^2 Processors, $O(1)$ Memory

Even when only $O(1)$ memory per PE is available, template matching can be done in $O(M^2)$ time. The algorithm repeatedly shifts the image and template

```
{Compute C2D using O(M) memory per PE }
{ Step1, compute C[q](i,j), 0 ≤ i,j < N, 0 ≤ q < M}
Accumulate(A,I,M); {each PE accumulates, in A, the next M template values}
for q := 0 to M−1 do
begin
  C[q](i,j) := 0, 0 ≤ i,j <N;
   for v := 0 to M−1 do
   begin
     PE (q,v) broadcasts T(q,v) to all PEs;
     B(i,j) := content(bus), 0 ≤ i,j < N;
     C[q](i,j) := C[q](i,j) + A[v](i,j) * B(i,j), 0 ≤ i,j < N;
   end;
end;
{Step2, compute C2D(i,j), 0 ≤ i,j < N }
ColumnAdjacentSum (C2D,C,M);
```

Fig. 14 N^2 processor, $O(M)$ memory, template matching

values so that each $PE\ (i,j)$ always has an image and template value whose product contributes to $C2D\ (i,j)$. The algorithm is given in Fig. 15. The initial and final configurations are the same as for the algorithm of Fig. 14.

```
C2D(i,j) := 0;
for u := 0 to M−1 do
{ compute terms involving T[u,*] }
begin
  A := I;
  for v := 0 to M−1 do
   { compute terms involving T[u,v] }
  begin
    PE (i,j) connects its N and W switches, 0 ≤ i,j < N;
    PE (u,v) broadcasts T(u,v);
    A(i,j) := content(bus), 0 ≤ i,j < N;
    C2D(i,j) := C2D(i,j)+ A(i,j) * I(i,j), 0 ≤ i,j <N;
    CircularRowShift (I,−1); {Shift left circularly by 1 }
  end;
  { restore I values }
  I := A;
  { set up I for next v}
  CircularColumnShift (I, 1); {shift up circularly by 1 }
end;
```

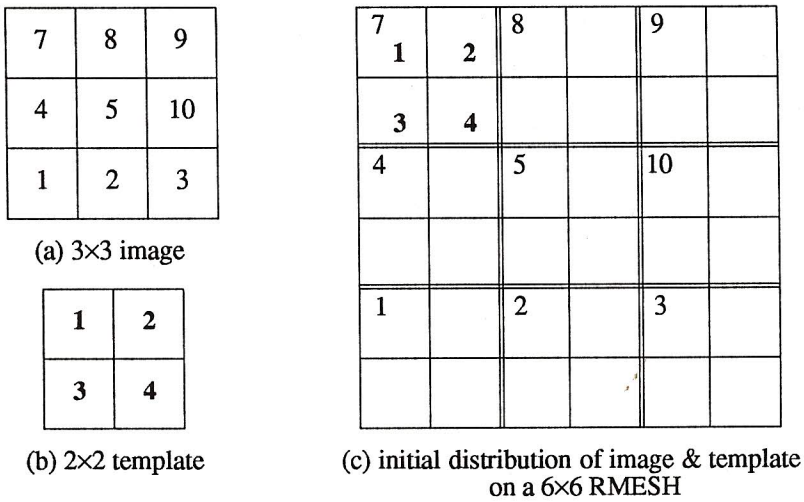
Fig. 15 N^2 PEs, $O(1)$ memory algorithm for $C2D$

While the asymptotic complexity of the $O(M)$ memory and $O(1)$ memory algorithms is the same, the $O(M)$ memory algorithm requires $M^2 + 3(M-1)$

broadcasts while the $O(1)$ memory one requires $4M^2 + 3M$.

6.3. N^2M^2 Processors, $O(1)$ Memory

We assume that the N^2M^2 processor RMESH is configured as an $NM \times NM$ array and that: initially, $I(iM,jM) = I[i,j]$, $0 \le i,j < N$ and $T(i,j) = T[i,j]$, $0 \le i,j < M$. The initial distribution of I for the case $N = 3$, $M = 2$ is given in Fig. 16. The final configuration will have $C2D(iM,jM) = C2D[i,j]$, $0 \le i,j < N$.



(a) 3×3 image

(b) 2×2 template

(c) initial distribution of image & template on a 6×6 RMESH

Fig. 16 Example initial configuration of an N^2M^2 RMESH with $N = 3$, $M = 2$.

The $NM \times NM$ processor array is naturally partitioned into N^2 $M \times M$ processor blocks as in Fig. 16 (c) (the partitions are of size 2×2 and are demarkated by double lines). The (i,j)'th such block is used to compute all the terms involving row i of the image that contribute to column j of $C2D$. For this, PE $(iM+q, jM+k)$ first gets an image value from its row such that $I(iM+q, jM+k) = I[i, (j+k) \bmod N]$ (step 2 of Fig. 17). This can be done in $O(1)$ time. The image values are initially in PEs (iM,jM), $0 \le i,j < N$. These locations are divided into M equivalence classes based on the value of $j \bmod M$. The values in each column can be broadcast to all PEs that need them using three broadcasts each. The first broadcast sends the image value in column j to row $j \bmod M$ of its $M \times M$ sub RMESH. The next uses row sub buses to broadcast to the submeshes that need the image values. The third broadcasts on column buses local to each sub RMESH. The template T is then broadcast, in step 3, to all $M \times M$ windows resulting in the configuration:

$$T(iM+q, jM+k) = T[q,k],\ 0 \le i,j < N,\ 0 \le k,q < M$$

This requires another M broadcast steps and uses the window broadcast algorithm of Section 2.2. Step 4 computes the product of I and T in each processor and requires no broadcasts. The result of this step is

$$C(iM+q, jM+k) = I[i, (j+k) \bmod M] * T[q,k],\ 0 \le i,j < N,\ 0 \le k,q < M$$

Step 1 Collect image values such that
$I(iM, jM+k) = I[i, (j+k) \bmod N],\ 0 \le i,j < N,\ 0 \le k < M$

Step 2 Broadcast the I values in row 0 of each $M \times M$ block along columns of the $M \times M$ block.
Following this, we have
$I(iM+q, jM+k) = I[i, (j+k) \bmod N],\ 0 \le i,j < N,\ 0 \le k,q < M.$

Step 3 Broadcast the template T to all $M \times M$ blocks.

Step 4 $C(a,b) = I(a,b) * T(a,b),\ 0 \le a,b < NM.$

Step 5 Sum the C's in each row of each $M \times M$ block to get

$$D(iM+q, jM) = \sum_{k=0}^{M-1} C(iM+q, jM+k),\ 0 \le i,j < N,\ 0 \le q < M.$$

Step 6 Broadcast D along rows of each $M \times M$ block to get

$$D(iM+q, jM+k) = \sum_{k=0}^{M-1} C(iM+q, jM+k),\ 0 \le i,j < N,\ 0 \le q,k < M.$$

Step 7 Let $D(iM+q, jM+k) = nil$ if $(i+M-q) \bmod M \neq k,\ 0 \le i,j < N,$
$0 \le q,k < M$

Step 8 Sum the non nil D values in each column in groups of size M to get $C2D$

Fig. 17 N^2M^2 processor algorithm

The next step is to sum the C values in each row of each $M \times M$ block to get:

$$D(iM+q, jM) = \sum_{k=0}^{M-1} C(iM+q,\ jM+k)$$

$$= \sum_{k=0}^{M-1} I[i, (j+k) \bmod N] * T[q,k]$$

This is done using the data sum operation which requires $O(\log M)$ broadcasts. We observe now that $D(iM+q, jM)$ contributes to $C2D[a,j]$ where $a = (i+M-q) \bmod M$). In fact,

$$C2D[a,j] = \sum_{(i+M-q) \bmod M = a} D(iM+q, jM)$$

To compute this sum efficiently, we assign column r of each $M \times M$ block the task of computing $C2D[a,j]$ for $a \bmod M = r$. For this, in step 6, we broadcast D values along rows of each $M \times M$ block and then is step 7 the D's not needed to compute the $C2D$'s assigned to a column are set to nil. Step 6 takes 1 broadcast and step 7 takes 0. The $C2D$ values can now be computed by summing the

non nil values in each column in groups of size M. This requires $O(\log M)$ broadcasts. The total complexity of the N^2M^2 processor algorithm is therefore $O(M)$.

The $O(M)$ complexity of the N^2M^2 algorithm is disappointing as this results in a processor-time product of $O(N^2M^3)$ which exceeds that of the serial algorithm by a factor of $O(M)$. However, using a data bandwidth argument we can show that $O(M)$ time is optimal given our initial configuration. Since the template is initially in the upper left $M \times M$ block of processors and the template is needed outside this block, M^2 pieces of data must flow out of the block. However, only $2M-1$ pieces of data can exit the block at any time as the block boundary includes only $2M-1$ processors. So, at least $M^2/(2M-1) = O(M)$ time is needed to broadcast the template to the rest of the RMESH.

7. Conclusions

The RMESH architecture (as well as its relatives such as the PARBUS, plymorphic torus, and MRN) are well suited for the solution of image processing problems. In many cases, the problems can be solved faster than on other proposed architectures. The RMESH algorithms are often conceptually simpler than the corresponding algorithms on other parallel architectures.

8. References

[1] D. H. Ballard and C. M. Brown, *Computer Vision*, Prentice Hall, New Jersey, 1985.

[2] Y. Ben-Asher, D. Peleg, R. Ramaswami, and A. Schuster, "The power of reconfiguration," *Journal of Parallel and Distributed Computing*, 13, 139-153, 1991.

[3] J. Chang, O Ibarra, T. Pong, and S. Sohn, "Two-dimensional convolution on a pyramid computer", Proceedings of the 1987 International Conference on Parallel Processing, The Pennsylvania State University Press, 1987, pp 780-782.

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

[5] Z. Fang, X. Li, and L. M. Ni, "Parallel algorithms for image template maching on hypercube SIMD computers", Proceedings of IEEE Workshop on Computer Architecture and Image Database Management, 1985, pp 33-40.

[6] K. S. Fu, *Syntactic Methods in Pattern Recognition*, Academic Press, New York, 1974.

[7] K. Fukunaga, *Introduction to Statistical Pattern Recognition*, Academic Press, New York, 1972.

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

[9] K. Hwang and D. Kim, "Parallel pattern clustering on a multiprocessor with orthogonally shared memory", Proceedings of the 1987 International Conference on Parallel Processing, The Pennsylvania State University Press, 1987, pp 913-916.

[10] J. Jang and V. Prasanna, "An optimal sorting algorithm on reconfigurable meshes," *Proceedings 6th International Parallel Processing Symposium*, IEEE Computer Society, Los Alamitos, CA, 130-137, March 1992.

[11] J. Jang, H. Park, and V. K. Prasanna, "A fast algorithm for computing histogram on reconfigurable mesh," *Proc. Frontiers of Massively Parallel Computing", 1992.*

[12] J. Jang, H. Park, and V. K. Prasanna, "An optimal multiplication algorithm on reconfigurable mesh," Department of EE-Systems, Technical Report IRIS-291,University of Southern California, LosAngeles, March 1992.

[13] J. Jang and V. K. Prasanna, "Efficient Parallel Algorithms for Some Geometric Problems on Reconfigurable Mesh," *Proceedings of 1992 International Conference on Parallel Processing*, CRC Press, Boca Raton, FL, 127-130, Aug 1992.

[14] J. Jenq and S. Sahni, "Reconfigurable mesh algorithms for image shrinking, expanding, clustering, and template matching," *Proceedings 5th International Parallel Processing Symposium*, IEEE Computer Society Press, Los Alamitos, CA, 208-215, 1991.

[15] J. Jenq and S. Sahni, "Reconfigurable mesh algorithms for the Hough transform," *Proc. 1991 International Conference on Parallel Processing*, CRC Press, Boca Raton, FL, 34-41, 1991.

[16] J. Jenq and S. Sahni, "Reconfigurable mesh algorithms for the area and perimeter of image components," *Proc. 1991 International Conference on Parallel Processing*, CRC Press, Boca Raton, FL, 280-281, 1991.

[17] J. Jenq and S. Sahni, "Histogramming on a reconfigurable mesh computer", *Proceedings International Parallel Processing Symposium*, 1992, 425-432.

[18] J. Jenq and S. Sahni, "Reconfigurable mesh algorithms for fundamental data manipulation operations", In *Parallel computing on distributed memory multiprocessors*, Ed. F. Ozguner, Springer Verlag, NATO ASI Series F, 1993.

[19] S. Lee and J. K. Aggarwal "Parallel 2-D convolution on a mesh connected array processor", Proceedings IEEE Conference on Computer Vision and Pattern Recognition, pp 305-310.

[20] X. Li and Z. Fang, "Parallel algorithms for clustering on hypercube SIMD computers", Proceedings IEEE Conference on Computer Vision and Pattern Recognition, 1986, pp 130-133.

[21] H. Li and M. Maresca, "Polymorphic-torus architecture for computer vision," *IEEE Trans. on Pattern & Machine Intelligence*, 11, 3, 133-143, 1989.

[22] H. Li and M. Maresca, "Polymorphic-torus network," *IEEE Trans. on Computers*, C-38, 9, 1345-1351, 1989.

[23] S. Lin, "Constant-time algorithms for the area and perimeter of image components on the processor arrays with reconfigurable bus systems", National Taiwan Normal University, 1992.

[24] P. MacKenzie, "A separation between reconfigurable mesh models", *Proceedings 1993 International Parallel Processing Symposium.*

[25] M. Maresca and H. Li, "Morphological operations on mesh connected architecture: A generalized convolution algorithm", Proceedings IEEE Conference on Computer Vision and Pattern Recognition, 1986, pp 299-304.

[26] 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.

[27] 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.

[28] 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.

[29] R. Miller, V. K. Prasanna Kumar, D. Reisis and Q. Stout, "Efficient parallel algorithms for intermediate level vision analysis on the reconfigurable mesh", *Parallel Architectures and Algorithms for Image Understanding*, Viktor K. Prasanna ed., 185-207, Academic Press, NewYork, 1991

[30] R. Miller, V. K. Prasanna Kumar, D. Reisis and Q. Stout, "Image processing on reconfigurable meshes", *From Pixels to Features II*, H. Burkhardt ed., Elsevier Science Publishing, Amsterdam, 1991.

[31] 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.

[32] D. Nassimi and S. Sahni, "Data broadcasting in SIMD computers",

[33] L. M. Ni and A. K. Jain, "A VLSI systolic architecture for pattern clustering", IEEE Transcations on Pattern Analysis and Machine Intelligence, vol. PAMI 7, no. 1 Jan. 85, pp 80-89.

[34] M. Nigam, and S. Sahni, "Constant time computational geometry on reconfigurable meshes with buses", University of Florida, Technical Report, 1992.

[35] M. Nigam, and S. Sahni, "Sorting n Numbers on $N \times N$ Reconfigurable Meshes with Buses, *International Parallel Processing Symposium*, 1993.

[36] S. Olariu, J. Schwing, and J. Zhang, "Data movement techniques on reconfigurable meshes, with applications", Old Dominion University, Technical Report, 1992.

[37] S. Olariu, J. Schwing, and J. Zhang, "Interval-related problems on reconfigurable meshes", Old Dominion University, Technical Report, 1992. *International Conference on Application-Specific Array Processors, 1992.*

[38] S. Olariu, J. Schwing, and J. Zhang, "Optimal convex hull algorithms on enhanced meshes", Old Dominion University, Technical Report, 1992.

[39] S. Olariu, J. Schwing, and J. Zhang, "Constant-time convex polygon algorithm on reconfigurable meshes", Old Dominion University, Technical Report, 1992.

[40] V. K. Prasanna Kumar and V. Krishnan, "Efficient image template maching on SIMD hypercube machine", Proceedings of the 1987 International Conference on Parallel Processing, The Pennsylvania State University Press, 1987, pp 765-771.

[41] S. Ranka and S. Sahni "Convolution on SIMD mesh connected multiprocessors", Proceedings 1988 International Conference on Parallel Processing, The Pennsylvania State University Press, 1988, pp 212-217.

[42] S. Ranka and S. Sahni "Image template maching on SIMD hypercube mutiprocessors", Proceedings of the 1988 International Conference on Parallel Processing, The Pennsylvania State University Press, 1988, pp 84-91.

[43] S. Ranka and S. Sahni "Image template maching on MIMD hypercube mutiprocessors", Proceedings of the 1988 International Conference on Parallel Processing, The Pennsylvania State University Press, 1988, pp 92-99.

[44] S. Ranka and S. Sahni "Hypercube algorithms for image transformations", Proceedings of the 1989 International Conference on Parallel Processing, The Pennsylvania State University Press, 1989, pp 24-31.

[45] S. Ranka and S. Sahni, *Hypercube algorithms with Applications to Image Processing and Pattern Recognition*, Springer Verlag, 1990.

[46] S. Ranka and S. Sahni, "Clustering on an SIMD hypercube multicomputer", *IEEE Trans. on Parallel and Distributed Systems*, 2, 2, 1991, 129-137.

[47] A. Rosenfeld and A. C. Kak, *Digital picture processing*, Academic Press, New York, 1982.

[48] A. Rosenfeld, "A note on shrinking and expanding operations in pyramids", Pattern Recognition Letters, 1987, vol. 6, no. 4, pp 241-244.

[49] J. Schwing and J. Zhang, "Fast component labeling on reconfigurable meshes", Old Dominion University, Technical Report, 1992.

[50] J. T. Tou and R. C. Gonzalez, *Pattern recognition principles*, Addison-Wesley, 1974.

[51] B. Wang and G. Chen, "Constant time algorithms for the transitive closure and some related graph problems on processor arrays with reconfigurable bus systems," *IEEE Trans. on Parallel and Distributed Systems*, 1, 4, 500-507, 1990.

[52] B. Wang, G. Chen, and F. Lin, "Constant time sorting on a processor array with a reconfigurable bus system," *Information Processing Letters*, 34, 4, 187-190, 1990.

[53] C. C. Weems, S. P. Levitan, A. R. Hanson, E. M. Riseman, J. G. Nash, and D. B. Sheu, "The image understanding architecture, " *International Journal of Computer Vision*, 2, 251-282, 1989.