Reconfigurable Mesh Algorithms For Image Shrinking, Expanding, Clustering, And Template Matching*

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

Abstract
Parallel reconfigurable mesh algorithms are developed for the following image processing problems: shrinking, expanding, clustering, and template matching. Our $N \times N$ reconfigurable mesh algorithm for the q-step shrinking and expansion of a binary image takes $O(1)$ time. One pass of the clustering algorithm for N patterns and K centers can be done in $O(MK + K\log N)$, $O(KlogNM)$, and $O(M + \log NMK)$ time using N, NM, and NMK processors, respectively. For template matching using an $M \times M$ template and an $N \times N$ image, our algorithms run in $O(M^2)$ time when N^2 processors are available and in $O(M)$ time when N^2M^2 processors are available.

Keywords and Phrases
reconfigurable mesh computer, parallel algorithms, image processing, shrinking, expanding, clustering, template matching.

* 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.

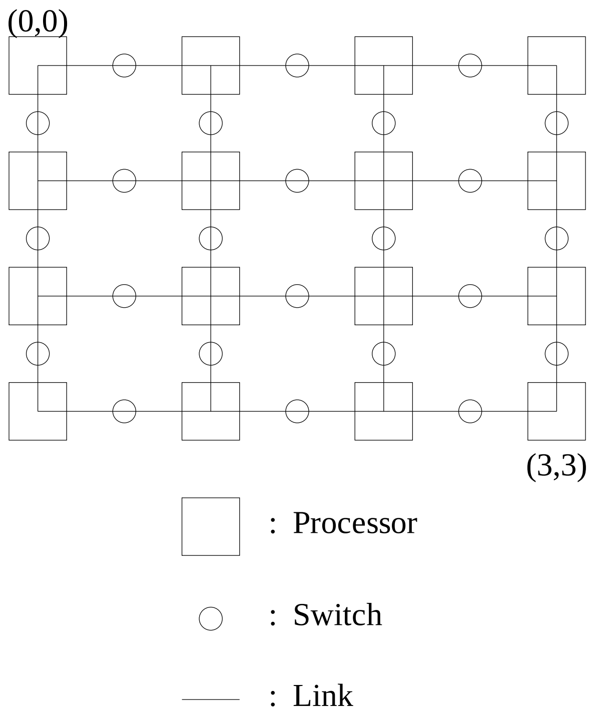

Figure 1 4×4 RMESH

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 some image processing problems. The specific problems we consider are: shrinking and expanding (Section 3), clustering (Section 4), and template matching (Section 5). In Section 2, we develop some basic data manipulation algorithms for the RMESH. These are used in subsequent sections to obtain the 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.

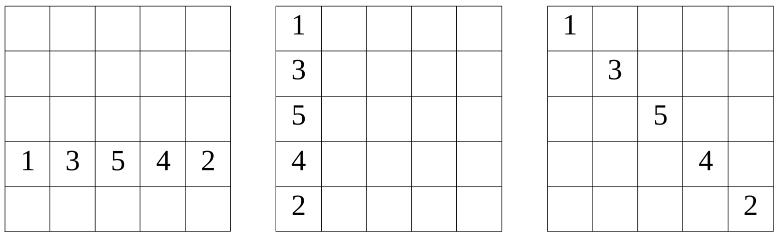

Figure 2 Diagonalize 4th row or 1st column elements of a 5×5 window

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 $A(i,j) = A(i\ mod\ w, j\ mod\ w)$ ($A(i,j)$ denotes variable A of PE (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 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 :

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

This can be done in $O(logN)$ time by first performing a prefix sum [MILL88] 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.

```
procedure WindowBroadcast(A,w);
{ broadcast the A values in the upper left w×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×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, $0 \le i < w$;
        $B(k, k \bmod w) :=$ content(bus), $0 \le k < N$;
        set switches to form row buses;
        $PE(k, k \bmod w)$ broadcasts its B value on its row bus, $0 \le k < N$;
        $A(k,i) :=$ content(bus) for $i \bmod w = j$, and $0 \le k < N$;
    end;
end;
```

Figure 3 Window broadcast

2.3 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

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

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.

2.4 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:

$$A[q](i,j) = I(i,(j+q) \bmod M)$$

This can be done using $2M - 1$ broadcasts. The algorithm is given in Figure 5.

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

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

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

```
procedure Accumulate (A,I,M)
{ each PE accumulates in A, the next M I values }
PE (i,j) disconnects its S switch and connects its W switch, $0 \le i,j < N$;
begin
        {accumulate from the right}
        for $k := 0$ to $M-1$ do
        begin
            {PEs (i,j) with $j \bmod M = k$ broadcast to PEs
            on their left that need their I value}
            PE (i,j) disconnects its E switch if $j \bmod M = k$
            and then broadcasts $I(i,j)$;
            $A[(k+M-(j \bmod M)) \bmod M](i,j) :=$ content(bus);
        end;
        {accumulate from the left}
        Each PE (i,j) disconnects its S switch and connects its W switch, $0 \le i,j < N$;
        for $k := 0$ to $M-2$ do
        begin
            PE (i,k) broadcasts $I(i,k)$, $0 \le i < N$;
            $A[q+k](i,N-q) :=$ content(bus), $1 \le q < M-k$;
        end;
end;
```

Figure 5 Data accumulation

Figure 6. 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
  $S(i,j) := X[((j \ mod \ M)+1) \ mod \ M](i,j), 0 \le i,j < N$;
  for k := 2 to M do
  begin
    {circularly shift S in 1×M blocks and add terms }
    RowCircularShift (S, M,-1)
    $S(i,j) := S(i,j) + X[((j \ mod \ M)+k) \ mod \ M](i,j), 0 \le i,j < N$;
  end;
end;
```

Figure 6 Consecutive sums in 1×M blocks

2.6 Adjacent Sum

We consider 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

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

Since the algorithms for both are similar, we discuss only the one for row adjacent sum. The strategy is similar to that for consecutive sum. Each processor initiates a token that will accumulate the desired sum for the processor that is $M-1$ units to its left. That is PE (i,j) initiates the token that will eventually have the desired value of $S(i, (N+j-M+1) \bmod N)$, $0 \le i,j < N$. The tokens are shifted left circulary 1 processor at a time until they reach their destination PE. The details of the algorithm are given in Figure 7. As each circular shift by -1 requires 3 broadcasts, the algorithm of Figure 7 requires $3(M-1)$ broadcasts.

```
procedure RowAdjacentSum (S,X,M);
begin
    $S(i,j) := X(i,j)[M-1]$;
    for $k := M-2$ down to 0 do
    begin
        RowCircularShift (S,N,-1);
        $S(i,j) := S(i,j) + X[k](i,j)$;
    end;
end;
```

Figure 7 Row adjacent Sum

3 Shrinking And Expanding

Let $I[0..N-1, 0..N-1]$ be an $N \times N$ image. The **neighborhood** of the pixel [i,j] is the set of pixels

$$nbd(i,j) = \{[u,v] \mid 0 \le u,v < N, \max\{|u-i|, |v-j|\} \le 1\}$$

The **q-step shrinking** of I is defined in [ROSE82] and [ROSE87] to be the $N \times N$ image S^q such that

$$S^1[i,j] = \min_{(u,v) \in nbd(i,j)} \{I[u,v]\},\ 0 \le i < N,\ 0 \le j < N$$

$$S^q[i,j] = \min_{(u,v) \in nbd(i,j)} \{S^{q-1}[u,v]\},\ q > 1,\ 0 \le i < N,\ 0 \le j < N$$

The **q-step expansion** of I is similarly defined to be the $N \times N$ image E^q where

$$E^1[i,j] = \max_{(u,v) \in nbd(i,j)} \{I[u,v]\},\ 0 \le i < N,\ 0 \le j < N$$

$$E^q[i,j] = \max_{(u,v) \in \, nbd(i,j)} \{ E^{q-1}[u,v]\}, \; q > 1, \; 0 \le i < N, \; 0 \le j < N$$

When the image is binary the max and min operators used above may be replaced by the logical **or** and **and** operators, respectively. Rosenfeld [ROSE87] present 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 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 [RANK89] 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.

Let $B_{2q+1}[i,j]$ represent the block of pixels:

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

So, $nbd(i,j) = B_3[i,j]$. Rosenfeld [ROSE87] has shown that

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

From this it follows that

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

where

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

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

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:

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

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

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

The steps involved in computing R^q for a binary image I are given in Figure 8. The complexity is readily seen to be $O(1)$. E^q is similary computed from R^q in $O(1)$ time. The algorithm of Figure 8 assumes that all switches are initially connected and that if a processor reads a bus and finds no value, the value ∞ is used.

4 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 [BALL85], [DUDA73], [FUKU72], [FU74], [ROSE82], and [TOU74]. 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

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

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 Figure 9.

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.

```
{ 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
          {Note that 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 $right(i,j)$
```

Figure 8 Computing *R* for a binary image

Hwang and Kim [HWAN87] have developed an algorithm for a multiprocessor with orthogonally shared memory; Ni and Jain [NI85] have proposed a systolic array; Li and Fang [LI86] have developed an $O(KlogNM)$ algorithm for an SIMD hypercube with *NM* processors; and Ranka and Sahni [RANK88a] have developed an $O(K + logNMK)$ algorithm for an SIMD hypercube with *NM* processors as well as an $O(logNMK)$ algorithm for an SIMD hypercube with *NMK*

```
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
```

Figure 9 One pass of the iterative cluster improvement algorithm

processors.

In this section we develop clustering algorithms for an RMESH. We consider RMESH's with, respectively, *N*, *NM*, and *NMK* processors. The time complexity of these algorithms is, respectively, $O(MK + KlogN)$, $O(KlogMN)$, and $O(M + \log NMK)$

4.1 *N* Processors

We assume that the *N* processor RMESH is configured as a $\sqrt{N} \times \sqrt{N}$ array. Initially each processor contains one row of the feature matrix and the *K* cluster centers are stored one per processor. For definiteness, assume that $F[q](i,j) = F[i\sqrt{N} + j,\ q]$, $0 \le i,j < \sqrt{N}$, $0 \le q < M$ (note that $F[q](i,j)$ denotes the *q*'th entry in the array *F* in processor (i,j) of the RMESH array of processors and $F[r,s]$ denotes an entry in the feature matrix *F*); $C[q](i,j) = C[i\sqrt{N} + j,\ q]$, $0 \le i\sqrt{N} + j < K$, $0 \le q < M$. The computation of the new cluster assignments (step 1 of Figure 9) can be done as in Figure 10.

In iteration *q* of the **for** loop the PE that contains cluster center *q* broadcasts it to all RMESH PEs. Following this, each PE computes the squared distance between this center and the pattern it contains. If this squared distance is smaller than the least squared distance computed so far for this pattern then cluster *q* becomes the candidate for the new cluster assignment

```
$D2(i,j) := \infty,\ 0 \le i,j < \sqrt{N}$
for $q := 0$ to $K-1$ do
begin
    $PE\ (i,j),\ i\sqrt{N} + j = q$ broadcasts $C[*](i,j)$ (i.e., cluster center $q$) to all PEs;
    Every $PE\ (a,b)$ reads the broadcast cluster center into its array
    $A[*](a,b)$ and then computes

        $X(a,b) := \sum_{s=0}^{M-1} (F[s](a,b) - A[s](a,b))^2$

    if $X(a,b) < D2(a,b)$ then
     [ $NewCluster(a,b) := q;\ D2(a,b) := X(a,b)$];
end;
```

Figure 10 N processor RMESH algorithm for cluster reassignment

of

this pattern. The correctness of Figure 10 is easily established. Each iteration of the **for** loop takes $O(M)$ time as the center broadcast involves the transmission of M values over a bus that includes all N processors and the computation of the squared distance involves $O(M)$ arithemetic per processor. The overall time for Figure 10 is therefore $O(MK)$. Step 2 of Figure 9 can be done in $O(1)$ time as in Figure 11.

The new cluster centers are computed one cluster at a time. To compute the new center of cluster i we need to sum, componentwise, the feature vectors of all patterns in cluster i and divide by the number of such patterns. The number of patterns in cluster i can be determined by having each PE set its A variable to 1 if its pattern is in this cluster and 0 if it isn't. Then the data sum operation of Section 2.2 can be used to sum up the A's in $O(logN)$ time. Let us therefore concentrate on computing the sum of the feature vectors of the patterns in cluster i. For this, we consider two cases $M \ge \sqrt{N}$ and $M < \sqrt{N}$.

If $M \ge \sqrt{N}$, the algorithm of Figure 13 is used. Steps 1 through 3 sum the feature vectors in

```
Step 1   if $NewCluster(a,b) = OldCluster(a,b)$
           then $mark(a,b) := false$
           else [$mark(a,b) := true$,
             $OldCluster(a,b) := NewCluster(a,b)$]
             $0 \le a,b < \sqrt{N}$
Step 2   PEs $(i, 0)$ for $i$ even disconnect their S switch
         PEs $(i, \sqrt{N}-1)$ for $i$ odd disconnect their S switch
         PEs $(i,j)$ for $j \notin \{0, \sqrt{N}-1\}$ disconnect their S switch
         This results in the bus of Figure 12
Step 3   if $mark(i,j)$ then PE $(i,j)$ disconnects its W switch in case $i$ is odd
         and its E switch in case $i$ is even, $0 \le i,j < \sqrt{N}$
          {Following this the bus of Figure 12 extends only up to
         the first PE on the bus with $mark(i,j) = true$}
Step 4   if $mark(i,j)$ then PE $(i,j)$ broadcasts 1 on its bus;
         PE [0,0] reads its bus;
         if a 1 is not read then PE (0,0) terminates the program;
```

Figure 11 N processor RMESH algorithm to update cluster assignments and terminate if necessary

cluster i along the rows of the RMESH. This sum is eventually stored in the column 0 processors. The strategy employed is quite similar to that used in Section 2.5 to compute consecutive sums. Step 4 then sums these sums and step 5 sends the overall sum to PE (0,0). Each step of Figure 13 other than step 2 takes $O(M)$ time. Step 2 takes $O(1)$ time. So, the overall complexity is $O(M)$.

If $M < \sqrt{N}$, tile the $\sqrt{N} \times \sqrt{N}$ RMESH with $1 \times M$ squares. Figure 14(a) shows the tiling for the case when M divides $\sqrt{N}$ and Figure 14(b) shows it for the case when M does not divide $\sqrt{N}$, *'s denote partial tiles. Consider, first, the case when M divides $\sqrt{N}$. Now the feature vector sums can be computed using the algorithm of Figure 15. The algorithm is itself explanatory. Steps 1 and 6 each take $O(M)$ time, steps 2 and 5 each take $O(logN)$ time; and steps 3 and 4 each

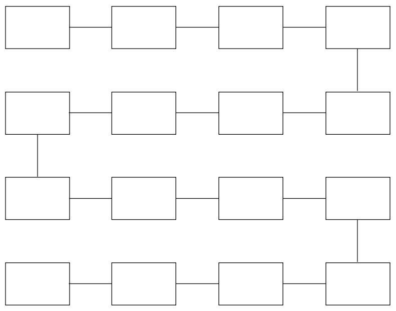

Figure 12 Bus resulting from switch setting of step 2 of Figure 11

take $O(1)$ time. The overall complexity of the algorithm of Figure 15 is therefore $O(M + logN)$

When $M < \sqrt{N}$ and M doesn't divide $\sqrt{N}$, the last two blocks of each row can be considered as a single block of length at most $2M-1$ (i.e., the *'d block in each row of Figure 14(b) is combined with the $1 \times M$ block immediately to its left). The algorithm is essentially that of Figure 15 except that the consecutive sum operation (step 1) is modified to handle the larger block at the end of each row. This modification still takes $O(M)$ time. Further, the right most $\sqrt{N}$ *mod* M columns of the RMESH are not involved in steps 2 through 6. The complexity remains $O(M + logN)$.

$O(M + \log N)$ time suffices to compute each cluster center. To compute all K of them takes $O(MK + KlogN)$ time. Since cluster reassignment and update take $O(MK)$ and $O(1)$ time, respectively, the time for one pass of the iterative cluster improvement algorithm is $O(MK + KlogN)$ on an RMESH with N PEs.

4.2 *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 Figure 16. Summing the E values takes $O(logM)$ time. The remaining steps each take $O(1)$ time. The overall complexity is therefore $O(KlogM)$.

Step 2 of Figure 9 (i.e., terminate and update) is easily done in $O(1)$ time. The cluster centers may be updated in $O(KlogN)$ time using the algorithm of Figure 17. The overall

---

Step 1 PE $(q, 0)$ initiates M tokens of type 1 one at a time.
These tokens travel along row q of the RMESH in a pipelined manner.
The j'th token accumulates the sum of the j'th feature value of all patterns in row q that are in cluster i, $0 \le q < \sqrt{N}$.

Step 2 Set up row buses.

Step 3 PE $(q, \sqrt{N}-1)$ broadcasts the tokens back to PE $(q, 0)$.

Step 4 PE $(0,0)$ initiates M tokens of type 2 one at a time.
These travel along column 0 of the RMESH.
The j'th token of type 2 accumulates the sum of the j'th token of type 1 in each of the column 0 processors.

Step 5 PE $(\sqrt{N}-1,0)$ broadcasts the M type 2 tokens it has received to PE (a,b), $a\sqrt{N}+b = i$.

Figure 13 Computing the feature vector sum for cluster i

---

complexity of one pass of the iterative cluster improvement algorithm is therefore $O(KlogMN)$ on an NM processor RMESH.

4.3 *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 $C(0,j) = C[j\ div\ M,\ j\ mod\ M]$, $0 \le j < KM$ initially. Our RMESH algorithm for cluster reassignment (Figure 18) 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 $O(logMK)$ time. Hence the complexity of the algorithm of Figure 18 is $O(M + \log MK)$. Step 2 of Figure 9 is easily done in $O(1)$ time.

The new cluster centers can be computed in $O(logN)$ 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 $C[j\ div\ M,\ j\ mod\ M]$. The complexity of one pass of the iterative improvement

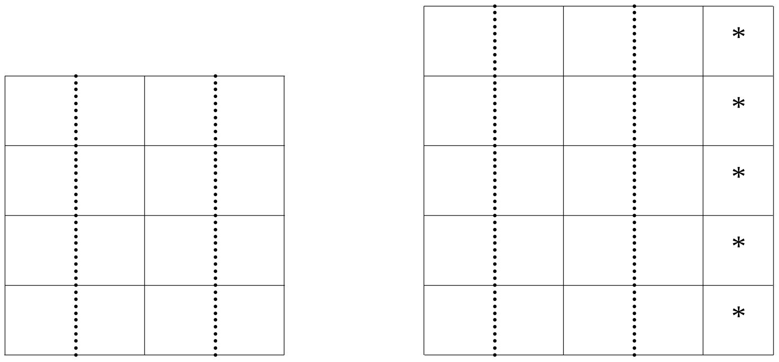

Figure 14 Tiling a $\sqrt{N}\times\sqrt{N}$ RMESH with $1\times M$ tiles

algorithm is therefore $O(M + logNMK)$ when NMK processors are available.

5 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:

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

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. [CHAN87] have developed an $O(M^2 + logN)$ algorithm for a pyramid computer with N^2 processors at its base. Ranka and Sahni [RANK88b] have developed an $O(M^2)$ algorithm for an N^2 processor mesh connected computer. Maresca and Li [MARE86] and Lee and Agarwal [LEE86] also develop mesh algorithms. Hypercube algorithms have been developed by Fang, Li and Ni [FANG85], Prassanna Kumar and Krishnan [PRAS87], and Ranka and Sahni [RANK88bcd]. The SIMD hypercube algorithm of [RANK88c] uses N^2 processors and has complexity $O(M^2 + \log N)$ and the MIMD

Step 1 The consecutive sum operation of Section 2.5 is used in each $1 \times M$ block of PEs . The data is the feature vectors in the block that correspond to patterns in cluster i. The result is stored in each PE's A variable. So, the PE in each $1 \times M$ block computes, in its A variable, the sum of the j'th feature vector of all patterns in its block that are also in cluster i.

Step 2 The A values are summed along the columns of the RMESH. The result is stored in the B variables of the processors in row 0.

Step 3 The B values are broadcast down column buses. Following this the B values in PEs in the same column are the same.

Step 4 $PE\ (a,b)$ sets its B value to 0 if $a > M-1$ or $a \neq b \ mod \ M$. Following this the nonzero B values in row a, $a < M$ all correspond to feature a.

Step 5 Sum the values on each row using the data sum operation of Section 2.2. Put the result in the D variable of column 0 processors.

Step 6 PEs $(a, 0)$, $0 \leq a < M$ broadcast their D values serially to $PE\,(c,d)$, $c\sqrt{N}+d = i$. $PE\ (c,d)$ saves the received M values as the sum of the feature vectors of cluster i.

Figure 15 Summing the feature vectors in cluster i

hypercube algorithm of [RANK88d] uses N^2 processors and has complexity $O(M^2)$. Ranka and Sahni [RANK88d] 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.

```
$D2(i, 0) = \infty,\ 0 \le i < N$
**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**;
```

Figure 16 *NM* processors algorithm for cluster reassignment

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.

5.1 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

```
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;
```

Figure 17 Computing cluster centers with *NM* processors

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

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

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

This is done by simply using the column adjacent sum operation of Section 2.6. The details are given in Figure 19. The complexity is $O(M^2)$.

5.2 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 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 Figure 20. The initial and final configurations are the same as for the algorithm of Figure 19.

```
Step 1  Create $K$ copies of $F$ so that $F(i,j) = F[i,j \ mod \ M]$,
        $0 \le j < KM$, $0 \le i < N$.
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
        $B(a,q) = \min\{B(a,j) \mid j \ mod \ M = 0\}$;
        store this $q$ in $NewCluster(a, 0)$;
```

Figure 18 New cluster determination with *NMK* processors

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 $4M^2 + 3M$.

5.3 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 Figure 21. The final configuration will have $C2D(iM,jM) = C2D[i,j]$, $0 \le i,j < N$.

The $NM\times NM$ processor array is naturally partitioned into N^2 $M\times M$ processor blocks as in Figure 21 (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 $I(iM+q,jM+k) = I[i,(j+k) \ mod \ N]$ (step 2 of Figure 22). 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 $j \ mod \ M$. The values in each column can be broadcast

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

Figure 19 N^2 processor, $O(M)$ memory, template matching

to all PEs that need them using three broadcasts each. The first broadcast sends the image value in column j to row j *mod* 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 2.1. Step 4 computes the product of I and T in each processor and requires no broadcasts. The result of this step is

```
$C2D(i,j)$ := 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, $0 \le i,j < N$;
    PE $(u,v)$ broadcasts $T(u,v)$;
    $A(i,j)$ := content(bus), $0 \le i,j < N$;
    $C2D(i,j) := C2D(i,j) + A(i,j) * I(i,j)$, $0 \le i,j < N$;
    $CircularRowShift(I,-1)$; {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;
```

Figure 20 N^2 PEs, $O(1)$ memory algorithm for $C2D$

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

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

$$\begin{aligned} D(iM+q, jM) &= \sum_{k=0}^{M-1} C(iM+q, \; jM+k) \\ &= \sum_{k=0}^{M-1} I[i, (j+k) \; mod \; N] * T[q,k] \end{aligned}$$

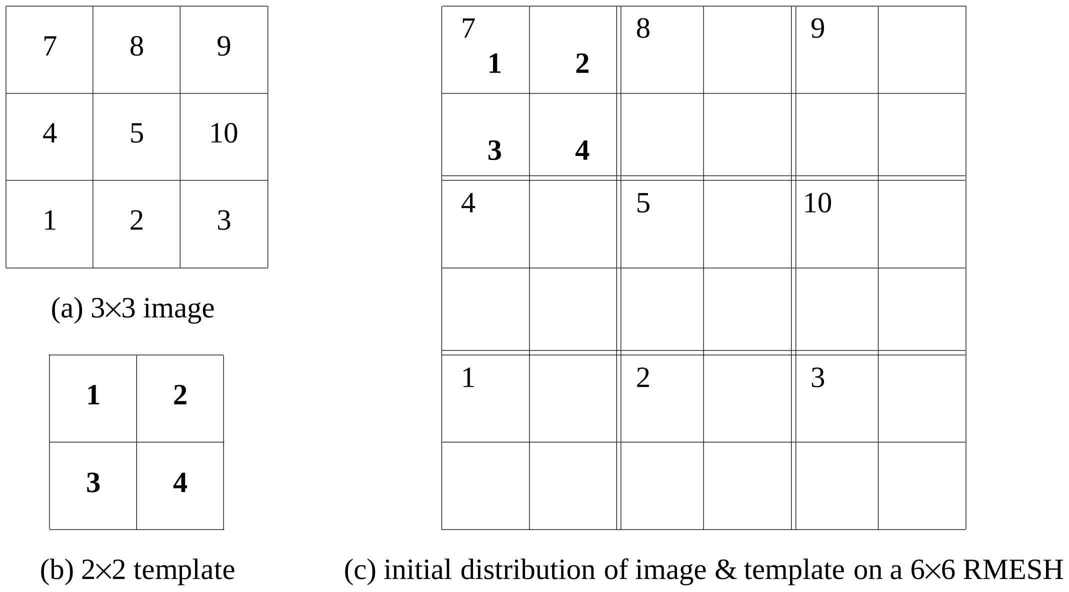

Figure 21 Example initial configuration of an N^2M^2 RMESH with $N = 3$, $M = 2$.

This is done using the data sum operation which requires $O(logM)$ broadcasts. We observe now that $D(iM+q,jM)$ contributes to $C2d[a,j]$ where $a = (i+M-q) \; mod \; M)$. In fact,

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

To compute this sum efficiently, we assign column r of each $M \times M$ block the task of computing $C2D[a,j]$ for $a \; mod \; M = r$. 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 $O(logM)$ 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

Step 1 Collect image values such that

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

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

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

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 $(i+M-q) \, mod \, M \ne k, \ 0 \le i,j < N,$ $0 \le q,k < M$

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

Figure 22 N^2M^2 **processor algorithm**

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.

6 Conclusions

We have developed parallel RMESH algorithms for shrinking, clustering, and template matching. Tables 1 - 3 compare the complexity of these algorithms with those for the hypercube and mesh connected computers.

q step expanding			
	Mesh	**Hypercube**	**RMESH**
binary	$O(q)$	$O(logq)$	$O(1)$
gray scale	$O(q)$	$O(logq)$	-

Table 1 N^2 processor image expanding

Clustering			
	Mesh	Hypercube	RMESH
N PEs	-	-	$O(MK + KlogN)$
NM PEs	-	$O(K + logNMK)$	$O(KlogNM)$
NMK PEs	-	$O(logNMK)$	$O(M + logNMK)$

Table 2 Clustering complexity

Template Matching			
	Mesh	Hypercube	RMESH
N^2 PEs	$O(M^2)$	$O(M^2 + \log N)$	$O(M^2)$
$N^2 M^2$ PEs	-	$O(logN)$	$O(M)$

Table 3 Template matching complexity

7 References

[BALL85] D. H. Ballard and C. M. Brown, *Coputer Vision*, Prentice Hall, New Jersey, 1985.

[CHAN87] 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.

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

[FANG85] 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.

[FU74] K. S. Fu, *Syntactic Methods in Pattern Recognition*, Academic Press, New York, 1974.

[FUKU72] K. Fukunaga, *Introduction to Statistical Pattern Recognition*, Academic Press, New York, 1972.

[HWAN87] 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.

[LEE86] 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.

[LI86] 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.

[MARE86] 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.

[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, 1988, 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.

[NI85] 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.

[PRAS87] 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.

[RANK88a] S. Ranka and S. Sahni "Clustering on A Hypercube Multicomputer", TR 88-15, University of Minnesota.

[RANK88b] 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.

[RANK88c] 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.

[RANK88d] 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.

[RANK89] 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.

[ROSE82] A. Rosenfeld and A. C. Kak, *Digital picture processing*, Academic Press, New York, 1982.

[ROSE87] A. Rosenfeld, "A note on shrinking and expanding operations in pyramids", Pattern Recognition Letters, 1987, vol. 6, no. 4, pp 241-244.

[TOU74] J. T. Tou and R. C. Gonzalez, *Pattern recognition principles*, Addison-Wesley, 1974.