Enhanced Mesh-Connected Computers for Image Processing Applications[1]

Mounir Hamdi
Department of Computer Science
Hong Kong University of Science and Technology
Clear Water Bay, Kowloon, Hong Kong
Email : hamdi@cs.ust.hk

Abstract

The Reconfigurable Array with Spanning Optical Buses (*RASOB*) has recently been introduced as an efficient enhanced mesh-connected parallel computer. *RASOB* combines some of the advantage characteristics of reconfigurable meshes and meshes with optical pipelined buses. In this paper, we use this computing model for the efficient design of fundamental image processing applications. In order to fully assess the potential of *RASOB*, the image processing applications chosen range from low-level processing (convolution, histogramming) where only neighborhood communications are need to high-level processing (connected component labeling, computational geometry) where global communications are needed.

1 Introduction

Numerous parallel architectures have been proposed for general purpose computing and in particular for image processing applications. Among the various proposed architectures, reconfigurable architectures are shown to be the most attractive. Reconfigurable architectures are attractive because they provide alternatives to completely connected systems at lower implementation costs. Since optical interconnects can offer many advantages over its electronic counterpart including high connection density and relaxed bandwidth-distance product, they will soon be a viable alternative for *multiprocessor interconnections* [2, 5]. This paper reviews the *Reconfigurable Array with Spanning Optical Buses* (*RASOB*) architecture that provides flexible reconfiguration as well as rich connectivities at low hardware and control complexities [13]. Then, we use this architecture for the efficient implementation of fundamental image processing algorithms.

A major difference between the *RASOB* architecture and other two dimensional array with either optical or electronic buses [3, 4] is that *there is a direct connection between any two processors.* More specifically, in a *RASOB*, a processor at row i and column j can send a message, without buffering and address decoding at any intermediate processor, to a processor at row k and column l, even if $i \neq k$ and $j \neq l$. Such a direct connection between these two processors at different rows and different columns can be established by setting an electro-optical switch [1] that interconnects the row i and column l. We will refer to the operation of setting switches as *hardware reconfiguration* in a *RASOB*.

Moreover, the *RASOB* architecture also takes advantage of the two important properties of the *optical transmissions*, namely, unidirectional propagation and predictable unit propagation delay. More specifically, the processors in a *RASOB* can be programmed to send and receive messages under synchronized control, such that a connection between a source and a destination is established by letting the source send a message at a specific point in time and letting the destination receive the message at another specific point in time [3, 6]. We refer to this type of reconfiguration as *software reconfiguration*. Because some of the reconfiguration is done in software, the complexity of both hardware and control required for reconfiguration can be kept low. However, despite of its low control and hardware complexity, the proposed *RASOB* architecture provides flexible reconfiguration that leverages the high communication bandwidths available in optical interconnects and is thus promising for efficient parallel implementation of many communication intensive algorithms.

The rest of the paper is organized as follows. Section 2 describes the *RASOB* architecture, in which both software and hardware reconfigurations are discussed. In section 3, some fundamental image processing applications are implemented on *RASOB* to high-light the efficiency of such architectures in the execution of these communication intensive applications. Finally, we conclude the paper in Section 5.

[1]This research work was supported in part by the Hong Kong Research Grant Council under the grant RGC/HKUST 100/92E.

0-8186-7134-3/95 $04.00 © 1995 IEEE

2 Architectural Model

The *RASOB* architecture is similar to the array structure described in [8]. A main difference is that in this architecture, messages are sent and received according to specific timing requirements. This makes this architecture suitable for SIMD applications. On the other hand, the structure in [8] employs an addressing mechanism which supports MIMD applications at higher hardware and control complexities. Figure 1 and Figure 2 illustrates the architecture of an *RASOB*. As shown in Figure 1, there are n folded row buses and n folded column buses interconnecting the processor array. Each processor has a transmitting interface to the upper segment of a row bus, and two receiving interfaces to the lower segment of the row bus and the right segment of a column bus, respectively.

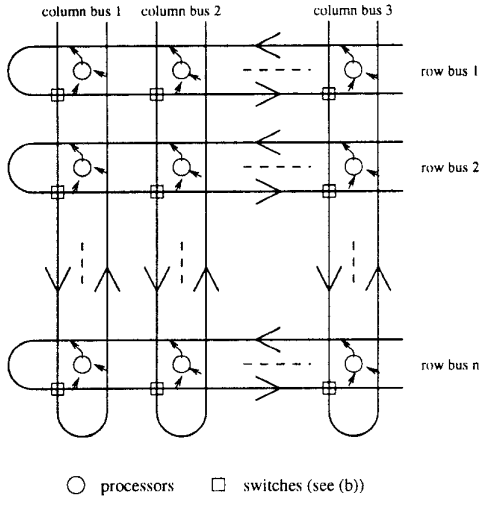

Figure 1: The architecture of *RASOB*.

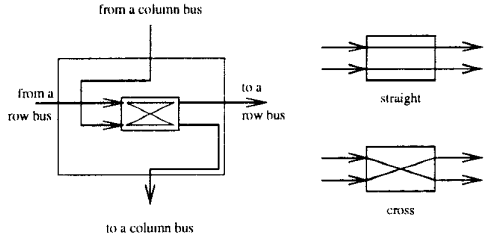

Figure 2: A switch interconnecting a row and a column bus.

A distinct architectural feature of the *RASOB* is that a 2×2 electro-optical switch is placed at the intersection of a row and a column bus, as shown in Figure 2. When the switch is set to straight, a message arriving along a row bus will continue propagating on the row bus; Otherwise, the message will be *switched* to the column bus instead. During a specific period, all the switches at a given row are set to *straight* and messages propagate only on a row bus. As a result, processors at a row communicate with each other at the same row. This type of communications is referred to as "Row communications" and the period during which row communications is accomplished is referred to as a *Row phase*.

A processor may also communicate with a processor at a *different row*, which may or may not be at a different column. This type of communications is referred to as "Column communications" and is accomplished by *switching* the message from a row bus to the desired column bus during a period called *Column phase*. In doing so, the switches are set to "cross" for the duration of the message and then changed back to the straight state.

Changing the state of the switches in a column phase is an example of what we called *hardware* reconfiguration. As a contrast, *software* reconfiguration in this paper refers to the programming of the processors so that they will send and receive messages at some specific points of time. In the following sections, we first illustrate software and hardware reconfiguration and then discuss both hardware and control complexities of an *RASOB* as well as its connectivities.

2.1 Software Reconfiguration

In a row phase, each row bus operates independently from the others so it is sufficient to describe just one row bus (e.g. row bus r), as shown in Figure 3. In the following presentation, we will denote the processors at row r from left to right by $p(r,1)$, $p(r,2)$, ... and $p(r,n)$, respectively.

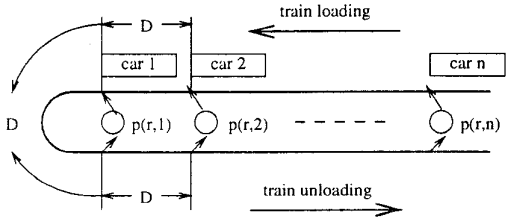

Figure 3: Train loading/unloading on a row bus

There are two important optical transmission properties, namely, *unidirectional propagation* and *predictable propagation delay* of the optical signals, that make concurrent access of an optical bus possible. More specifically, with an appropriate spatial separation between the neighboring processors, message collision can be avoided even when the processors are transmitting messages concurrently [3, 6]. In the following discussions, we assume that each processor on a row bus is separated in time by $D = bw + \delta$ (seconds) from its neighbors, where b is the the maximal length of a packet in bits, w is the optical pulse width (or bit duration) in seconds, and $\delta > 0$ is used as *guard bands* to tolerate synchronization error to a certain degree.

This temporal separation can be achieved by separating the two neighboring *transmitter interfaces* on the upper segment as well as the *receiver interfaces* on the lower segment of a row bus with a fiber length $D \times c$, where c is the speed of light in the fiber, as shown in Figure 3. Without loss of generality, we assume that the length of the folded part, which is the separation of the transmitter and receiver interface of $p(r,1)$, is also made equivalent to D.

We may use the train loading/unloading model to describe the operations in a row phase. Let us imagine that at the beginning of a row phase, a *train* (or motorcade) of n cars is originated at the rightmost end of the upper segment of the row bus. Each car can be regarded as an empty packet slot with a duration of D and is numbered 1 through n from left to right. During a row phase, the switches that connect the row bus with column buses are in the "straight" state so that the train will run through the lower segment of the row bus. A simple assignment of the cars is to let processor $p(r,1)$ use car 1 for sending its packet, let $p(r,2)$ use car 2 for sending its packet and so on.

With this assignment of the cars, the time when $p(r,i)$ may transmit its packet, relative to the beginning of the row phase, is given by

$$RowSend[(r,i)] = (i-1)D+(n-i)D = (n-1)D \quad (1)$$

As a result, all processors will be transmitting simultaneously because the transmitting time does not depend on i. In addition, a receiving processor can determine the exact time when the car carrying the packet will arrive at its receiver interface. More specifically, if processor $p(r,i)$ is expecting a packet sent by $p(r,j)$, it can calculate the time it should pick up the packet as below,

$$\begin{aligned} RowRec[(r,i) \leftarrow (r,j)] &= (n-1)D + (i+j-1)D \\ &= (n+i+j-2)D \quad (2) \end{aligned}$$

By placing all the processors under a synchronized control and let each processor send and receive at specific points of time as in Eqs. 1 and 2, the row bus can be reconfigured into a variety of interconnection patterns.

2.2 Hardware Reconfiguration

If a processor needs to communicate with another processor at a different row, it has to send a packet in a column phase. The train loading/unloading model used previously is also useful in illustrating the principles involved in column communications. More specifically, we will let car 1 of the train make a *turn*, from the lower segment of a row bus, onto column bus n, car 2 make a turn onto column bus $(n-1)$, and so on. For simplicity, we assume that the switches are placed near the receiver interfaces so that the propagation delay between a switch and its nearby receiver is almost negligible. This also implies that the switches are placed D apart from each other.

Similar to Eq. 2, we can determine the time that car k arrives at switch $(n-k+1)$ to be

$$SwitchArvl[(r,n-k+1) \leftarrow (r,k)] = (2n-1)D \quad (3)$$

Since the right side of the equation does not contain k, every car arrives at its turning point at the same time. Therefore, one may set the switches on a row bus to "cross" simultaneously and by doing this, the n packets in the train are switched onto their respective destination columns, one packet per each column. This arrangement implies that during a Column phase, *two or more processors at the same row can not send packets destined to the same column.*

If $p(i,j)$ needs to communicate with $p(r,k)$ where $r \neq i$, $p(i,j)$ will have to transmit (or load) a packet into car $(n-k+1)$. We can determine the time for $p(i,j)$ to transmit its packet to be

$$\begin{aligned} ColSend[(i,j) \rightarrow (r,k)] &= (n-k)D + (n-j)D \\ &= (2n-j-k)D \quad (4) \end{aligned}$$

By separating the adjacent row buses by D, a column bus will look like a row bus that is turned 90^o anti-clockwise after the packets are switched. More specifically, that every row bus switches a packet onto a column bus at the same time is similar to the case where packets are transmitted to a row bus simultaneously. With as much as D separation between every two row buses, there will be again a train of n cars, each carrying a packet, formed on the left segment of a column bus. Hence, we can determine the time for $p(r,k)$ to pick up the packet at the its receiver interface on the column bus, which is sent by $p(i,j)$, to be

$$ColRec[(r,k) \leftarrow (i,j)] = (2n+i+r-2)D \quad (5)$$

2.3 Connectivity and Complexity

Software reconfiguration can be performed with little control overhead because each of the above equations (1 to 5) involves simple arithmetic calculations. In addition, the hardware complexity of the proposed architecture is low because each processor uses only one two-state 2×2 switch and has only one transmitter. Although two receiver interfaces are needed by each processor, a single high-speed electronic receiving circuit may be shared among these two interfaces. As a comparison, most mesh-based reconfigurable architectures would require at least an equal number of switches having four or more states and four I/O interfaces per each processor [7].

Despite of the low control and hardware complexities, the *RASOB* provides strong connectivities due to the following characteristics. First, a direct connection between any two processors can be established. Second, reconfiguration is flexible as one may *interleave* Row and Column phases in many ways to provide communication bandwidth required by an application. Finally, because only a portion of optical power is tapped

off at each receiver interface, *multicasting* can be supported simply by programming multiple receivers to receive at different points of time during the same phase. Noting that one-to-one and broadcast are special cases of multicasting, we may summarize the communication capability of the *RASOB* below:

- All processors at row i can multicast to the processors at the same row at the same time; and such an row-to-row multicasting can be performed at all n rows simultaneously.
- All processors at column j can multicast to the processors at the same column at the same time; and such an column-to-column multicasting can be performed at all n columns simultaneously.
- $p(i,j)$ can multicast to several processors at any column k and such a processor-to-column multicasting can be performed by all the n^2 processors at the same time.

Note that while the first two items on the list alone mean that the *RASOB* is at least as powerful as any mesh with row and column buses, the third item clearly shows that the *RASOB* has a stronger connectivity than other meshes with row and column buses.

3 Parallel Image Processing Algorithms

Although the *RASOB* has a strong connectivity, it, like many practically scalable architectures, has a weaker connectivity than a completely-connected network. The capabilities as well as restrictions of the architecture makes it an interesting yet challenging task to design efficient algorithms. In this Section, we present the design and analysis of various fundamental algorithms in image and vision processing. Image and vision computing is usually modeled as a three-stage process: low-level, intermediate-level, and high-level. In the following subsections, we will present two-dimensional convolution and histogram operations (low-level), connected component labeling (intermediate-level), and extreme point identification and diameter computation (high-level).

3.1 Convolution

Convolution is frequently used in various image processing applications including in the design of linear filters (applied in noise cleaning and quality enhancement) as well as in edge detection. The two-dimensional convolution $y_{i,j}$ can be expressed through the formula:

$$y_{i,j} = \sum_{s=o}^{k-1} \sum_{t=0}^{k-1} W_{s,t} x_{i-s,j-t}$$

where k^2 is the size of the convolution kernel $W(k \times k)$ and k is odd.

Two-dimensional convolution has been shown to be optimal on meshes [14]. M. Maresca and H. Li had also shown the mesh embedding in polymorphic torus architecture [15]. The algorithms they used can also be applied in *RASOB*, which takes $O(k^2)$ communication phases and $O(k^2)$ computation time. However, these algorithms do not fully utilized the connectivity of *RASOB*. In order to make a better utilization, we propose another algorithm here, which takes $k+1$ communication phases, and with $O(k^2)$ computation time on each processor.

We assume that the image X of size $n \times n$ pixels is stored in the *RASOB* of size $n \times n$, with each processor storing one pixel. Initially, the convolution kernel W of size $k \times k$ is stored in an array-form inside each PEs.

The algorithm of convolution is presented as follows:

1. Each PE sends its element to its $(k-1)/2$ left neighboring PEs and to its $(k-1)/2$ right neighboring PEs on the same row as illustrated in Figure 4.
2. Each PE now have k elements. For each PE, broadcast all k elements to its neighboring $(k-1)/2$ upper PEs and to its $(k-1)/2$ lower PEs in the same column.
3. Each PE performs the computation locally (to perform the above formula).

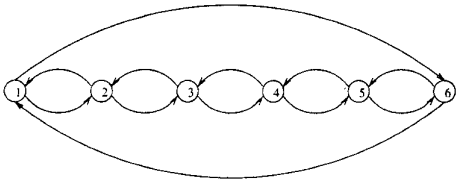

Figure 4: Step 1 of the algorithm in case of $n = 6$ and $k = 3$.

Theorem 1 *The convolution can be done using $k+1$ communication phases.*

Proof: It is not difficult to prove the above theorem. Step 1 can be done by one *row* phase. In step 2, each PE has to send k elements. So it takes k *column* phases. Step 3 involves local computation, which it can be easily seen that the formula requires k^2 additions. Thus, $k+1$ communication phases and $O(k^2)$ computation time are required.

3.2 Histogramming

The histogram of an image is a one-dimensional array $h[i], i = 0, \ldots, L-1$, such that $h(i)$ is the number

of pixels with value i. Histogram algorithms had been well developed in mesh, reconfigurable mesh, tree or hypercube. The algorithm on $RASOB$ is similar to the algorithm implemented on a reconfigurable mesh, which takes $O(L \log n)$ time [15].

The algorithm works as follows. If processor $p(x, y)$ contains the value i, it makes a signal of 1; otherwise, makes a signal of 0. Then, perform a sum on all the signals from the PEs will get the $h[i]$ values as detailed below.

Algorithm

```
for(i = 0; i < L; i++)
{
        for each PE
                if (value == i)
                        VOTE = 1;
                else
                        VOTE = 0;
        h[i] = sum(VOTE);
}
```

Theorem 2 *The time complexity of histogramming on an RASOB is* $O(L \log n)$.

Proof: The complexity is followed by the fact that the complexity of performing a sum on the whole $RASOB$ is $O(\log n)$. Executing the loop L times gives a $O(L \log n)$ time of the whole histogram algorithm.

3.3 Connected Component Labeling

Connected component labeling in digital images takes as an input an array of pixels $X_{i,j}$ (where each pixel can be binary, gray level, or color), with $i, j = 1, 2, 3, \ldots, n$, and produce as an output an array of labels $L_{i,j}$, such that all the elements of L correspond to a connected component if they are directly connected or if they are directly connected to a pixels belonging to the same connected component. Two pixels are *directly connected* if they are adjacent [i.e., at address (i, j) and $(i, j \pm 1)$ or (i, j) and $(i \pm 1, j)$] and have the same value (gray level or color). The labels are assigned to the connected components according to the following rule: each pixel is given a label corresponding to the *least* address (in row major order) of the pixels belonging to its component. Originally, each PE has the label of its own address.

Some research on the implementation of connected component labeling on Polymorphic torus architecture has been shown in [9], which takes $O(\log n)$ time complexity. However, this complexity carries a large constant term. A second one was proposed in [15], with a smaller constant term, but with a $O(\log^2 n)$ time complexity. The best one so far is [11], which takes constant time. This algorithm takes two advantages of reconfigurable bus networks. The first one is that each PE can set the connection of its four ports to its four neighboring PEs, and hence it can break down the whole architecture in any kinds of subbuses as it requires. Next, each PE can read all bus lines through its input buffers, and can write on the bus whenever that port is connected. Such technical features are not found on $RASOB$, and thus cannot be directly applied.

In this section, we proposed a $O(\log n)$ time complexity algorithm for connected component labeling on $RASOB$, with a small constant and some local computation. The idea of our algorithm follows that of the mesh algorithm suggested in [10], with time complexity $O(n)$ for a $n \times n$ image on $n \times n$ PEs. The improvement of the time complexity on $RASOB$ is due to its strong connectivity.

The algorithms follows a bottom-up, divide-and-conquer strategy in that it applies the same procedure to larger and larger regions, starting from single pixel regions and increasing the size up to the entire image. Pairs of regions are combined to form larger regions. The combination process happens both horizontally and vertically. Vertical combinations merge two square regions of size $h \times h$ to form a rectangle of size $2h \times h$, whereas horizontal combinations merge two rectangles of size $2h \times h$ to form a square of size $2h \times 2h$ as shown in Figure 5.

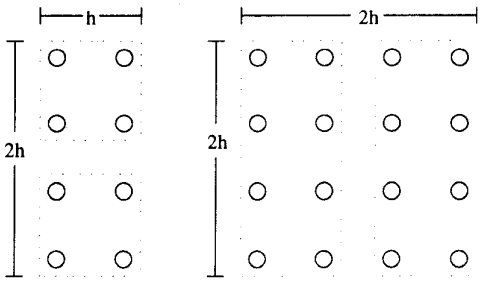

Figure 5: Image partitioning in connected component labeling.

The algorithm starts by examining all the pixels in parallel. Single pixels are considered square regions of size 1 and therefore the first combination is vertical, to form a rectangle of size 2×1. Then, horizontal and vertical combinations repeat alternatively until the entire image is reached. In each combination, either horizontal or vertical, two procedures will be carried out, namely *Combine()* and *Update()*. The *Combine()* and *Update()* routines are described as follows.

Combine():

For each pair of PEs crossing the boundary of each region to be merged, if they have the same value (gray level or color), a packet is created in both PEs. The packet is in the form of (*large-label, small-label*) where *large-label* is the label of the PE with a larger label, and *small-label* is the label of the other PE.

If they do not have the same value, they do noth-

ing, i.e., create *no* packet.
/* end of *Combine()* */

So far, it is checked completely for any connected component crossing the boundary. We need now to update the labels to the whole region, by the packet just created in the *Combine()* routine:

Update(): (for vertical combination)

1. Each PE on the boundary will now broadcast its own packet (if any) to all PEs on the same column in its own region as illustrated by Figure 6.

   /* Now, all PEs in the same column of the same region will contain the same packet. */

2. Each PE broadcasts its own packet to all other PEs in the same row of the same region.

   /* Now, each PE will receive at most k *different* packets, where k is the length of the boundary. */

3. Locally, each PE checks its own label against the packets it just received by the following rule:
   for each *different* packet *(first-label, second-label)* received,
   if (own-label == first-label)
   then
   own-label = second-label;

/* end of *Update()* */

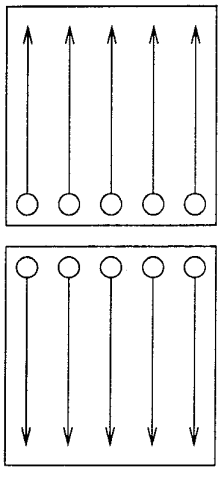

Figure 6: Directions of broadcasts to cover the whole region.

The case of horizontal combination is similar, just change the row phase to *column* and the *column* to *row*.

Now, the whole region is updated, thus can start another combination of a larger area.

After one vertical and one horizontal combination, the size of region will be multiplied by 4, i.e. from $h \times h$ to $2h \times 2h$. The number of such iteration is $\log n$. $(h = 1, 2, 4, 8, \ldots, n)$.

Thus, the whole pseudocode of the algorithm can be presented as follows:

```
for(h = 1; h < lgn; h++)
{
        /* Vertical Combination */
        Combine();
        Update();
        /* Horizontal Combination */
        Combine();
        Update();
}
```

Theorem 3 *Connected component labeling in an RASOB can be done in $O(\log n)$ time.*

Proof: *Combine()* procedure involves only one comparison for each PE on the boundary, and it requires one communication phase (either *column* or *row*, depending on *vertical* or *horizontal* combination). On the other hand, *Update()* procedure requires one column phase for step 1 and one *row* phase for step 2 for *vertical* combination. (In case of *horizontal* combination, one *row* phase for step 1 and one *column* phase for step 2). Notice that this can be done simultaneously for each region. Step 3 only involves local computations. The worst case of the number of local computation (comparison between own-label and label in packet received) in this step is k. However, in most images, adjacent pixels usually has the same value, hence adjacent packets in the boundary usually are the same, which greatly reduced the number of *different* packets in each combination. Besides, some of the pairs on the boundary may not have the same value, thus create no packet. Thus, the number of computation will usually be less than k.

Therefore, 3 communication phases are required in each combination, with at most k computations, where k is the length of the boundary at that combination. One vertical and one horizontal combination together form one iteration. $\log n$ iterations are required to reach the entire image. Thus, in conclusion, $6 \times \log n$ communication phases and at most n computations are required for the whole connected component labeling algorithm on an *RASOB*.

3.4 Extreme Point Identification

The extreme points of a set of processors S are the corners of the smallest convex polygon containing S. They basically delimit the convex hull of the set; $p(i, j)$ is an extreme point of S if $hull(S) \neq hull(S - p(i, j))$

Our algorithm described below follow closely the extreme point identification algorithm for Polymorphic torus [15]. The input to the algorithm is a binary

image $X_{i,j}$, with $i,j = 1,2,\ldots n$, stored one pixel per processor in an $n \times n$ *RASOB*. At the end of the algorithm, $p(1,m)$ is equal to 1 if $p(1,m)$ is an extreme point of S. The algorithm is divided in two sections, the first is for the left extreme points (LEP), situated on the left side of S, and the second is for the right extreme points (REP), situated on the right side of S. We will examine only the first section, concerned with the LEPs, considering that the REP section can be organized in an analogous way.

The first part of the algorithm for finding LEPs is to identify all possible left extreme points (PLEPs), which are the leftmost PEs having value 1 in each row. The second part is to discard those PLEPs which are not *true* left extreme points. The method to do this is based on some mathematical theories. Here, we just give some basic ideas. It is done by computing, for each $PLEP_i$, of row i, the angles $A_{i,j}, i = 0,1,\ldots,n-1$, between the horizontal axis and the lines connecting $PLEP_i$ and $PLEP_j$, as shown in Figure 7. For each $PLEP_i$, the maximum and the minimum among $A_{i,j}, j = 0,1,\ldots,n-1$ are then found and their difference is computed. If the difference is less than or equal to 180 degrees, then $PLEP_i$ is a true LEP; otherwise, it is discarded, as shown in Figure 8.

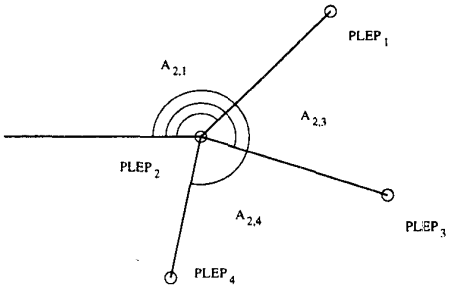

Figure 7: Computation to discard PLEPs

Before presenting the algorithm for finding LEPs, let us introduce a subroutine, *find-PLEP()*, which identify all possible left extreme points (PLEP).

Lemma: The subroutine *find-PLEP()* can be implemented using just one row communication phase.

Proof: It can be done as follows. For each PE having a 1 in row i, sends a signal to all PEs on its right hand side in row i. The one which has an 1 but not receiving any signal is the PLEP_i.

Now, we can present the algorithm for *left* extreme point identification.

1. Identify all PLEPs.
2. All PEs corresponding to PLEP_i broadcast their coordinates to all processors in row i.
3. Each diagonal processor $(p(i,i), i = 1,2,\ldots,n)$ broadcasts the coordinates just received to all the processors in the same column.
4. All $p(i,j)$ which contain the coordinates of both PLEP_i and PLEP_j compute the angle $A_{i,j}$ locally for all i,j (i.e. for all PEs).
5. Find the maxima and minima among the $A_{i,j}$ for each j over all the i, and store the result in $p(n,j)$.
6. Perform a subtraction between the two values just received in $p(n,j)$ locally.
7. The result is sent to the original position where $PLEP_j$ has been found.

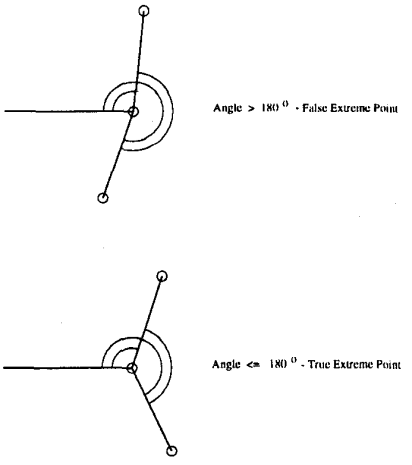

Figure 8: Example of true and false extreme points

Theorem 4 *The overall complexity of* left *extreme point identification in RASOB is* $O(1)$.

Proof: The proof is straight forward. Step 1 takes one row phase, by calling the subroutine *find-PLEP()*, as stated in the *lemma*. Step 2 takes one row phase, and step 3 takes one column phase. Step 5 takes constant time. Step 6 can be done by one column phase. Notice that after step 6, $p(n,j)$ will contain the result of whether $PLEP_j$ is a true LEP or not. To sent back the results to the PE containing $PLEP_j$, one easy way is the following. For each j, $p(n,j)$ broacasts its result to all PEs in row j. The PE having $PLEP_j$ will know the result, while the other PEs can ignore it.

Identifying the *right* extreme points can be done in the same way. Thus, the overall complexity of extreme point identification is $O(1)$, as stated above.

3.5 Diameter Computation

The diameter of a component is defined as the maximum distance between two of its extreme points. For one component, there are at most $2\sqrt{n}$ extreme points ($\sqrt{n}$ LEPs and $\sqrt{n}$ REPs). Thus, we have to

compute at most $4n$ distances and then find the maximum among them. In the following, we will present an $O(1)$ time algorithm for diameter computation.

Algorithm

1. Identify all extreme points; now, each row i will have at most two *active* PEs, which contains the LEP_i and REP_i.
2. The two *active* PEs in row i broadcast their coordinates to all PEs in row i.
3. $P(i,i)$ boardcasts the coordinates it just received *(at most two)* to all PEs in column i.
4. Each PE computes the distances among all extreme points locally and finds the maximum.
5. Find the global maximum.

Theorem 5 *Diameter computation on RASOB takes $O(1)$ time.*

Proof: The proof is also very straight forward. By the extreme point identification algorithm introduced in last section, step 1 can be done in constant time. Step 2 requires one row phase, and step 3 takes *at most* two column phases.

After step 3, $p(i,j)$ holds at most two LEPs and two REPs: LEP_i, LEP_j, REP_i and REP_j. Thus, step 4 involves at most 6 local computations for each PE. Step 5, of course, takes constant time. The global maximum found in this step will be the diameter of the component.

Thus, we can find the diameter of a $n \times n$ component on a $n \times n$ *RASOB* in $O(1)$ time.

4 Conclusion

The *reconfigurable array with optical spanning buses* (*RASOB*) architecture is anticipated to become one popular parallel architecture due to its unique properties of optical transmission to achieve flexible reconfiguration and strong connectivities with a low hardware and control complexity. In this paper, we took advantage of the unique and powerful properties of this architecture for the design and analysis of various fundamental image processing applications. The algorithms have been chosen to span a wide range of communication demands to fully assess the potential of *RASOB*. It was shown that most algorithms perform much better on an *RASOB* than on traditional reconfigurable electronic meshes. This is mainly due to the pipelining capabilities of the *RASOB*.

References

[1] R. Alferness, L. Buhl, S. Korothy, and R. Tucker. Highspeed Δ β-reversal directional coupler switch. In *Photonic Switching, OSA Technical Digest*, volume 13, pages 77-78, 1987.

[2] A. Guha, J. Bristow, C. Sullivan, and A. Husain. Optical interconnections for massively parallel architectures. *Applied optics*, 29(8), 1980.

[3] A. Guo et al. Array processors with pipelined optical busses. *Journal of Parallel and Distributed Computing*, 12(3):269-282, 1991.

[4] V.P. Kumar and C.S. Raghavendra. Array processors with multiple broadcasting. *Journal of distributed computing*, 2:173-190, 1987.

[5] P. Lalwaney, L. Zenou, A. Ganz, and I. Koren. Optical interconnects for multiprocessors: cost performance analysis. In *Proc. on Frontiers of Mass. Para. Comp.*, pp 278-285, Oct. 1992.

[6] R. Melhem, D. Chiarulli, and S. Levitan. Space multiplexing of waveguides in optically interconnected multiprocessor systems. *The Computer Journal*, 32(4):362-369, 1989.

[7] R. Miller, V.K.P.Kumar, D.I. Reisis, and Q.F. Stout. Parallel computations on reconfigurable meshes. *IEEE Trans. Comp.*, 42:678-692, 1993.

[8] C. Qiao and R. Melhem. Time-division optical communications in multiprocessor arrays. *IEEE Trans. Comp.*, 42(5):577-590, May 1993.

[9] M.C. Sheng and H. Li. Connected Conponent Labeling Algorithm on Polymorphic Torus. *Proc. of Int'l Computer Symposium 1988, Taipei, Taiwan*, pp.440-443, Dec 1988.

[10] M. Maresca, H. Li and M Lavin. Connected Component Labeling on Polymorphic Torus Architecture, In *ICPP*, pp.951-956, 1988.

[11] H. M. Alnuweiri. Parallel Constant-Time Connectivity Algorithms on a Reconfigurable Network of Processors. *IEEE Trans. on Parallel and Distr. Sys., vol. 6, No.1*, pp.105-110, Jan 1995.

[12] S. S. Lin, Constant-Time Algorithms for the Channel Assignment on Processor Arrays with Reconfigurable Bus Systems. *IEEE Trans. on Computer-Aided Design, July 1994*

[13] C. Qiao, Efficient Matrix Operations in a Reconfigurable Array with Spanning Optical Buses. *5th Symposium Frontiers of Massively Parallel Computations*, pp. 273-280, 1994.

[14] M. Maresca and H. Li. Morphological Operations on Mesh Connected Computers. *IEEE Intl. Conf. Computer Vision and Pattern Recognition*, pp. 299-304, 1986.

[15] M. Maresca and H. Li. Polymorphic VLSI Arrays with Distributed Control. *Reconfigurable Massively Parallel Computers*, by H. Li and Q.F. Stout.