

The Systolic Reconfigurable Mesh

MARY M. ESHAGHIAN-WILNER*

*Department of Electrical Engineering, University of California at Los Angeles
Los Angeles, CA 90095-1594, USA*

and

RUSS MILLER

*Department of Computer Science, State University of New York at Buffalo
Buffalo, NY 14260*

Received (November, 2003)

Revised (July, 2004)

Communicated by (J.L. Gaudiot)

ABSTRACT

In this paper, we introduce the *Systolic Reconfigurable Mesh (SRM)*, which combines aspects of the reconfigurable mesh with that of systolic arrays. Every processor controls a local switch that can be reconfigured during every clock cycle in order to control the physical connections between its four bi-directional bus lines. Data is input on one side of the systolic reconfigurable mesh and output from another side, one row/column per unit time. Efficient algorithms are presented for intermediate-level vision tasks, including histogramming, connectivity, convexity, and proximity.

Keywords: Reconfigurable mesh, image algorithms, component labeling, mesh-connected computer, and proximity.

1. Introduction

Over the past few decades, several mesh-based parallel architectures, including the mesh-connected computer, mesh-of-trees, and pyramid, have been considered for performing several interesting low- and intermediate-level computer vision tasks (*c.f.* [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16]). This is due to the fact that a two-dimensional image can be mapped in a straightforward fashion onto a two-dimensional mesh. In particular, reconfigurable meshes have been shown to be attractive computational engines relative to the traditional mesh-connected computer due to the flexibility that the reconfigurable bus offers in terms of long distance communication. Further, reconfigurable meshes are practical in that they can be implemented using VLSI technology [17,18,19,20,21,22], and can also be integrated with MEMS designs [23]. For a recent study of reconfigurable computing systems, refer to Bondalapati's thesis and related publications [24,25,26].

*The present address of the author.

The reconfigurable mesh was originally proposed as a massively parallel computing model in the late-1980s [27,28,12,29]. A review of the algorithmic literature [30] suggests that a major area of emphasis has been on *i*) fundamental problems, including sorting and arithmetic, *ii*) problems involving regularly structured data, such as matrices and images, in areas such as graph theory and image processing, and *iii*) geometric problems, where the amount of input data is sparse compared to the size of the reconfigurable mesh. Even though aspects of the reconfigurable mesh have been incorporated into the communication of the MasPar machines [31] and the Gated Connection Network of the Image Understanding Architecture [6], most of the algorithmic results in this area are of purely theoretical interest.

In this paper, we consider problems in low- and intermediate-level image processing. These problems include histogram, convex hull, nearest neighbor, and component labeling, to name a few. Given an $n \times n$ digitized black/white image, distributed in a natural fashion one pixel per processor on a mesh of size n^2 , efficient mesh algorithms have been constructed to solve these, and related problems, in $O(n)$ time [32]. While the mesh algorithms to solve these problems on $n \times n$ image input requires $\Omega(n)$ time, the running time of our algorithms to solve these problems on a systolic reconfigurable mesh of size n^2 is a function of the $O(n)$ input time, the $O(n)$ output time, and either $\Theta(1)$ or $\Theta(\log n)$ intermediate processing time.

We introduce a practical, scaled-down variant, of the reconfigurable mesh. The intent is to provide a computing engine that will significantly improve the performance over a large set of related algorithms on a realizable 21st century architecture. The rest of the paper is organized as follows. In the next section, we review the reconfigurable mesh and introduce the systolic reconfigurable mesh. In Section 3, we present efficient algorithms to solve a variety of related problems, predominantly in the area of intermediate-level image processing. Finally, in Section 4, we conclude by discussing a number of open problems using this novel model.

2. The Architecture

The *systolic reconfigurable mesh (SRM)* is a variation of the standard *reconfigurable mesh (RM)* models. Therefore, we first define the reconfigurable mesh and then extend this definition to that of the systolic reconfigurable mesh.

2.1. RM

The *mesh with reconfigurable bus*, or *reconfigurable mesh (RM)* of size n^2 consists of an $n \times n$ array of processors connected to a grid-shaped reconfigurable broadcast bus, where each processor has a locally controllable bus *switch*, as shown in Figure 1. The switches allow the broadcast bus to be divided into subbuses, providing smaller reconfigurable meshes. For a given set of switch settings, a *subbus* refers to a broadcast bus over a maximally connected subset of processors. Except for the buses and switches, the reconfigurable mesh is similar to the standard mesh. It is worth noting that, like the mesh, the reconfigurable mesh of size n^2 occupies $\Theta(n^2)$ area, under the assumption that processors, switches, and a link between adjacent switches occupy unit area. Further, notice that from a theoretical point

of view, there is no need to include the basic NEWS (north, east, west, and south) connections in the model, as the reconfigurable bus can be used to provide these connections in $\Theta(1)$ time by cycling through a simple set of four switch settings.

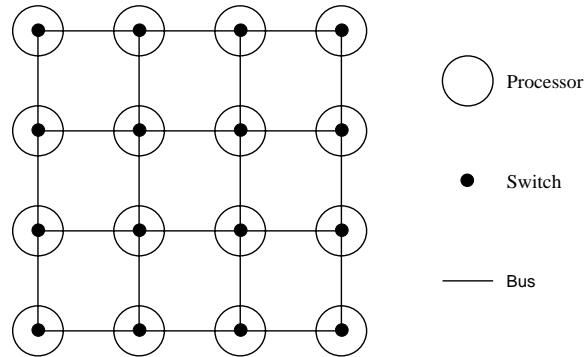


Fig. 1. A reconfigurable mesh of size 16.

While variations in the reconfigurable mesh model exist in terms of number of processors, layout of the processors, processing power, communication primitives, and so forth, a reasonably generic set of parameters includes the following.

- An *exclusive write*, in which one processor per disjoint subbus is permitted to broadcast a word of data to its subbus, which is available to all processors on the subbus.
- A *concurrent write*, in which multiple processors are permitted to simultaneously broadcast a single bit of data to their subbus, where the logical OR of the bits is available to all processors on the subbus.
- A *unit-delay broadcast*, in which all broadcasts take $\Theta(1)$ time.

Therefore, in $\Theta(1)$ time on the generic reconfigurable mesh model, every processor can perform a fixed number of arithmetic and Boolean operations on the contents of its own local memory, every processor can set the physical communication pattern between its four bus lines, either an exclusive write or a concurrent write can be performed on every subbus, and every processor can receive information sent to its subbus.

2.2. SRM

A *systolic reconfigurable mesh of size n^2* consists of n^2 processors arranged in a two-dimensional grid overlaid with a grid-shaped reconfigurable bus. See Figure 2. Every processor controls a local switch that can be reconfigured during every clock cycle to control the physical connections between its four bi-directional bus lines, realizing any of 15 possible combinations. Without loss of generality, assume that data is input from the left side of the systolic reconfigurable mesh, one column of

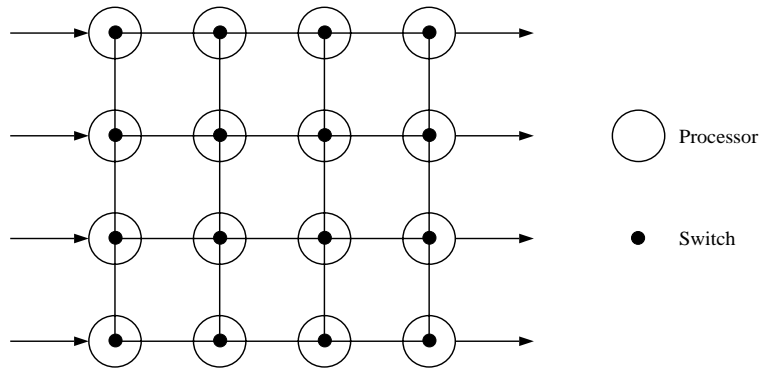


Fig. 2. A systolic reconfigurable mesh of size 16. Each generic processor contains a switch that is under local control and can be used to configure the four bus lines in any of 15 possible combinations. Input is from the left and output is to the right.

data per unit time, and eventually output from the right side, again, one column per unit time. Note that it is possible to design a systolic reconfigurable mesh, where the input comes from left but it is output from the top (see Algorithm 2 of section 3.5).

Given mn input items to be processed on a systolic reconfigurable mesh of size n^2 , the computations on the systolic reconfigurable mesh fall naturally into three phases. The first phase can be classified as the *input and preprocessing phase*, where at time t , n data items are input and a (small) fixed number of operations are performed over some subset of the tn items that have been input thusfar. After completing the input and preprocessing phase following time m , the *static computing phase* is performed on the $O(mn)$ items currently available in the systolic reconfigurable mesh. Finally, the *output and postprocessing phase* consists of pumping the data out, typically n items per unit time, and performing a (small) fixed number of operations on the remaining data, until all of the output has been produced. Whenever possible, it is desirable to design an algorithm with the property that it does not have a static computing phase and that the total running time is kept to a minimum.

Similar to the reconfigurable mesh, variations of the systolic reconfigurable mesh are possible. In this paper, we consider two such variations. They include the

1. *bit model (BM)*, where individual processors can operate on a fixed number of bits of data in unit time, and the
2. *word model (WM)*, in which individual processors can operate on a fixed number of $2 \log n$ bit words of data in unit time.

For both variations, we assume that concurrent writes of a single bit (for BM) or word (for WM) of data to a subbus is permitted if the values are identical. We also

assume that in unit time a bit (for BM) or word (for WM) of data can be broadcast to a subbus and read by all processors attached to that subbus. These assumptions are all reasonable and realizable in terms of designing and implementing a VLSI chip or a board that can be used in a workstation environment. Furthermore, every step of an algorithm designed for the word model can be simulated on a bit model variant in $O(\log n)$ time, where a word is $\log n$ bits wide.

3. Designing Algorithms

When designing algorithms for meshes or reconfigurable meshes, it is typically assumed that the data already resides in the processors. When designing algorithms for the systolic reconfigurable mesh, this assumption no longer holds, as the time to perform input and output of data is considered in the complexity analysis of an algorithm. Denote a reconfigurable mesh of size n^2 by R , and a systolic reconfigurable mesh of size n^2 by S . Suppose algorithm A requires t time on R . Then A can be simulated on S by first inputting the n^2 pieces of data in n unit-time steps, then performing algorithm A on S in t time, then outputting the $O(n^2)$ final entries in n unit-time steps. Therefore, the total time for simulating reconfigurable mesh algorithm A on a systolic reconfigurable mesh is $2n + t$. Suppose algorithm B requires t' time on a mesh of size n^2 , where $t' = \Omega(n)$. Then, the total time for simulating mesh algorithm B on a systolic reconfigurable mesh is $2n + t'$. Note that in the above two cases, the time complexities of the static phase are t and t' , respectively.

Therefore, if one were only concerned with an asymptotic analysis of algorithms on the systolic reconfigurable mesh, then one could simply simulate any mesh or reconfigurable mesh algorithm in $O(n)$ time. (Notice that the class of $O(n)$ time reconfigurable mesh algorithms properly includes the class of $O(n)$ time mesh algorithms.) However, in this paper we focus on practical algorithms that can be efficiently performed on a realizable architecture, such as a companion board in a workstation targeted at image processing applications. Therefore, our goal is to design algorithms that not only eliminate the static phase, but which are complete in no more than $2n$ cycles. More generally, we are concerned with

1. minimizing constants,
2. preserving the systolic nature of the systolic reconfigurable mesh, and
3. eliminating the static stage of the algorithms.

In this section, we present some fundamental algorithms for the systolic reconfigurable mesh. For convenience and consistency, we assume that the input to the problems is a set of n^2 data items fed into a systolic reconfigurable mesh of size n^2 .

3.1. Histogram

Consider the problem of creating a histogram of input values, where all such values are in the range of $1 \dots n$. The results will be maintained in the last column of the systolic reconfigurable mesh. Specifically, the total number of occurrences of value i , $1 \leq i \leq n$, will be maintained in $SRM(i, n)$. In this section, we present an algorithm that works for the word model and is easily adaptable to the bit model.

For the word model, every step takes $O(1)$ time, while for the bit model, every step takes $O(\log n)$ time.

The following is an outline of the word model/bit-model algorithm. It describes the operations performed during every cycle in the input and preprocessing phase once the new column of data has been input into the first column of the SRM. Again, assume that processor $SRM(i, 1)$ has just received a data item with value d .

1. Broadcast a single bit=1 along row i to $SRM(i, i)$.
2. Broadcast a single bit=1 from $SRM(i, i)$ to $SRM(d, i)$.
3. $SRM(d, i)$ adds the 1 received to its counter value, therefore it's counter is incremented by 1.

The three steps of the input and preprocessing cycle outlined above, take $o(1)$ time for the word model and $O(\log n)$ time for the bit model. There is no static computing phase in this algorithm. In the output and postprocessing phase, in every cycle, every processor except those located in the last column, passes the value of its counter in lockstep to its right neighbor. Then every right neighbor that receives this value adds it to its current count. Every cycle in this phase takes $O(1)$ time in the word model and $O(\log n)$ time in the bit model. The final results of the histogram will be ready as soon as the image leaves the mesh. The results will be stored in the last column, where the processor in the d th row of the last column has the count of occurrences of the value d in the image.

Theorem 1 *The histogram problem for n^2 input items in the range of $1 \dots n$ can be solved on the word-model (bit-model) systolic reconfigurable mesh of size n^2 using only a small fixed ($\Theta(\log n)$) number of broadcasts per input and preprocessing cycle, eliminating the static phase, and outputting the results upon completion.*

3.2. Convex Hull

Consider the problem of marking the extreme points representing the convex hull of an $n \times n$ binary image. While the results for the histogram problem just presented were stored in a convenient predetermined location of the SRM, for this problem the results will be maintained along with the image. That is, the value of a pixel at position (i, j) will be maintained as part of a record that also includes a flag indicating whether or not pixel (i, j) is an extreme point of the image. It should be noted that this "marked" flag will be set to true after the completion of cycle $t < n$ (i.e., after t columns of the image have been input and preprocessed) if and only if pixel (i, j) is an extreme point of the restriction of the image to the rightmost $n \times t$ subimage.

Given the word model, each input and preprocessing cycle consists of the following. Similar results have appeared in literature, see [33].

1. In lockstep, shift the image to the right while inputting the next column of data into the first column of the SRM.

2. Perform bus splitting in column one of the *SRM* for the purpose of identifying the topmost and bottommost black pixels that were just input.
3. Mark these pixels as extreme points and broadcast their coordinates to all processors.
4. Every marked pixel uses the coordinates of these two new extreme points, its point, and its previous extreme point in enumerated order, to determine whether or not it is to remain marked as an extreme point.
5. The extreme points that precede these two new extreme points in the enumerated ordering are identified and their locations are broadcast to these new extreme points.

Notice that as the data flows from left to right in a truly systolic fashion, the marked field and preceding point location field for each pixel remains tied to the pixel data. Therefore, once all of the data has been input, the data may be output in a natural fashion. Again, it is important to note that the static stage is nonexistent. Further, the output and postprocessing stage is reduced simply to outputting the highlighted/marked image.

Theorem 2 *Given an $n \times n$ binary image and a word-model (bit-model) systolic reconfigurable mesh of size n^2 , using only a small fixed ($\Theta(\log n)$) number of broadcasts per input and preprocessing cycle, eliminating the static phase, and outputting the image in lockstep fashion with no additional work, the convex hull of the image can be marked.*

3.3. Nearest Neighbor

Consider the problem of determining for every black pixel in an $n \times n$ binary image, the coordinates of a nearest black pixel. As with the previous problem, the input image will be fed into the *SRM* in a natural fashion, one column per input and preprocessing cycle, and for each pixel, the coordinates of a running nearest neighbor will be coupled to the pixel. During the output and postprocessing stage, when pixel (i, j) is output, so will the coordinates representing a nearest neighbor for pixel (i, j) . (Notice that while the distance to a nearest neighbor is necessarily unique, there may be multiple near neighbors. In the case where multiple nearest neighbors exist, the algorithm will record one such neighbor.)

The algorithm makes use of the triangle inequality. Given the word model, each input and preprocessing cycle consists of the following steps.

1. In lockstep, shift the image to the right while inputting the next column of data into the first column of the *SRM*.
2. Use bus-splitting and broadcasting, as well as communication between adjacent processors in the first column of the *SRM*, to determine for every pixel (whether black or white) in the first column, the nearest black pixel in the column both above and below.
3. Perform row broadcasts so that every processor $SRM(i, j)$ learns the coordinates of the two black pixels determined by processor $SRM(i, 1)$.

4. Every processor that contains a black pixel of the image determines whether either of these pixels is closer to its pixel than the previously recorded nearest neighbor, and if so, updates its nearest neighbor entry.

As with the previous algorithm, the data flows from left to right in a truly systolic fashion, and the desired information remains associated with the input data throughout the process. Therefore, once all of the data has been input, the desired information may be output in a natural fashion. Again, notice that the static stage is nonexistent and the output and postprocessing stage consists simply of outputting the required information as an image.

Theorem 3 *Given an $n \times n$ binary image and a word-model (bit-model) systolic reconfigurable mesh of size n^2 , using only a small fixed ($\Theta(\log n)$) number of broadcasts per input and preprocessing cycle, eliminating the static phase, and outputting the image in lockstep fashion with no additional work, a nearest neighboring black pixel can be identified for every black pixel in the image.*

3.4. Minimum or Maximum

Consider the problem of finding the minimum (or maximum) value from a set of n^2 input values. Notice that this problem differs significantly from the histogram problem previously presented in that there is no restriction on the range of input values. The algorithm is straightforward based on the word model. The details are not shown as similar work has appeared in the literature [34].

Theorem 4 *Given an $n \times n$ binary image and a word-model (bit-model) systolic reconfigurable mesh of size n^2 , using only a small fixed ($\Theta(\log n)$) number of broadcasts per input and preprocessing cycle, eliminating the static phase, and outputting a single value, the minimum or maximum value can be determined.*

3.5. Component Labeling

Consider the problem of labeling an $n \times n$ binary image. A simple approach would be to input the image in a natural fashion, apply a straightforward bit-polling algorithm to label the image [27], and then output the image in a natural fashion. Notice that the time for such an algorithm is $2n + \log_2 n$ and that 2 context switches are required due to the inclusion of the static stage. It is possible to overlap the static stage with the input stage by performing bit polling during every input step. This results in the image being labeled by the end of the input stage, requiring $n \log_2 n$ time. The labeled image can then be output by repeating the previous steps for another n steps (leading to a total running time of $2n \log_2 n$) or by context switching to n simple shift steps (leading to a total running time of $n \log_2 n + n$). Similarly, given an $n \times n$ adjacency matrix input, the $\log n$ bit polling algorithm of [6] can be applied to every input step so that in $n \log_2 n$ time the image is labeled. It becomes quite a challenge to produce an efficient algorithm that omits the static stage and also avoids repeating the $\log n$ -time based labeling schemes in each step.

In this section, we present three algorithms, all of which consider an $n \times n$ digitized image as input. The first one is a simple three phase algorithm that works for the word-model and well as bit model, and runs in $3n$ cycles, each cycle take $O(1)$

time in the word model and $O(\log n)$ time in bit model. The second algorithm is a two phase algorithm for the bit model, which runs in an optimal $2n$ cycles, where each cycle includes a $\log n$ time bit polling routine. Note that this algorithm receives the input from the left, but outputs the data from the top row of processors instead of from the right column of processors. The third algorithm runs in an optimal $2n$ cycles for the word-model. However, in each of the n output steps, the algorithm includes a constant time sorting operation, which is not as desirable in practice as it is in theory.

For the first two algorithms presented in this section, the component labels assigned to the pixels will be of the form $\langle C_R, C_L, R_T \rangle$, where C_R is the index of the rightmost column containing a pixel of the component, C_L is the index of the leftmost column containing a pixel of the component, and R_T is the index of the topmost row containing a pixel in the component.

3.5.1. Algorithm 1:

As with some of the previous algorithms, the component label record will remain coupled to the pixel value as the pixel travels through the systolic reconfigurable mesh. When a pixel is initially input to the SRM, the fields of the component label will be initialized to $\langle 0, 0, 0 \rangle$. These values will be updated throughout the course of the algorithm. The following is an outline of a straightforward component labeling algorithm for the word and bit model.

1. At time i , shift the entire image to the right while inputting column $n - i + 1$. Create a connected bus over every connected region. Next, every processor in column 1 of the SRM that contains a black pixel broadcasts the value $n - i + 1$ to be stored as C_L for its connected region. (Notice that after cycle $t = n$, all n columns have been input and C_L has been properly determined for every connected component.)
2. For $t = n + i$, for row $i = 1$ to n do: Every processor in row i that contains an upper boundary black pixel (white neighbor to its top) broadcasts the value i to be stored as R_T for its connected component. This value is stored in the appropriate field by a processor representing a black pixel only if $R_T = 0$. (Notice that after cycle $t = 2n$, the values of R_T have been properly determined.)
3. At cycle $2n + i$, the i^{th} column of the image is output as follows: For each processor in column i that contains a black pixel, if $C_R = 0$ then broadcast the value of i as C_R to all processors containing black pixels in its connected region, including itself; Shift out. (Notice that after cycle $t = 3n$, the n^{th} slice will be output, and all pixels that are output will have their correct final labels.)

Theorem 5 *Given an $n \times n$ digitized image, it can be labeled on the word-model (bit-model) systolic reconfigurable mesh of size n^2 using only a small fixed $(\Theta(\log n))$ number of broadcasts per cycle, and outputting the image in lockstep fashion, totalling $3n$ cycles including a static phase.*

3.5.2. Algorithm 2:

The algorithms we have presented thusfar take the input from the left and output the results to the right. However, in this algorithm, the SRM takes its input from the left and produces output to the top.

1. At time i , shift the entire image to the right while inputting column $n - i + 1$. Create a connected bus over every connected region. Next, every processor in column 1 of the SRM that contains a black pixel broadcasts the value $n - i + 1$ to be stored as C_L for its connected region. If $C_R = 0$ for a component receiving a broadcast, then set C_R to C_L . If $C_R \neq 0$ for a component, then use bit polling to find the minimum of the currently stored C_R 's over the connected region and store that value as C_R ; (After cycle $t = n$, the n^{th} column has been input, and C_L and C_R have been determined for every connected component.)
2. During cycle $n + i$, the i^{th} row of the image will be output from top as follows: All processors in row i containing an upper boundary black pixel (white neighbor to its top) with the value of $R_T = 0$, broadcasts the value i to be stored as R_T in their connected regions; Shift the image to the top. (After cycle $t = 2n$, the n^{th} row-slice is output to the top with the correct final labels.)

Theorem 6 *Given an $n \times n$ digitized image, it can be labeled on the bit-model systolic reconfigurable mesh of size n^2 using only $\Theta(\log n)$ number of broadcasts per input and preprocessing cycle, eliminating the static phase, and outputting the image in lockstep fashion from the top.*

3.5.3. Algorithm 3:

This algorithm is quite different from those that have appeared earlier in this paper in that there is a nontrivial input and preprocessing phase, as well as a nontrivial output and postprocessing phase. The component label will be represented as $\langle C_L, C_R, T \rangle$, where C_L is the index of the leftmost column containing a pixel of the component, C_R is the index of the rightmost column containing a pixel of the component, and T is used to break ties in the case of distinct figures with the same C_L and C_R values. C_L will be computed for every pixel during the input and preprocessing phase of the algorithm. C_R will be computed for every pixel during the output and postprocessing phase of the algorithm. Since a number of figures can occupy the same leftmost and rightmost column of the image, for such figures, a tie-breaking scheme will be used to determine T during the output and postprocessing step.

Each input and preprocessing cycle consists of the following.

1. In lockstep, shift the image to the right while inputting the next column of data into the first column of the SRM.
2. Initialize the component labels for all pixels now in column one of the SRM to $\langle 0, 0, 0 \rangle$.

3. All processors that currently hold a black pixel of the image, connect their bus to all neighboring pixels that also maintain black pixels. The result is that there is a subbus over all figures with respect to the restriction of the image that has thusfar been input.
4. Exploiting the concurrent write capability, all processors in the first column of the SRM now broadcast their column (with respect to the entire image) label. Note that during cycle c , this is column $n - c + 1$. All processors receiving such a value, store this in C_L , potentially replacing a previous value.

After n of these simple input and preprocessing cycles, all pixels know C_L , the leftmost column of any pixel in their connected component. Both C_R and T will be determined during the output and postprocessing phase. Each output and postprocessing cycle is concerned with those processors maintaining pixels in the last column of the SRM that have not previously received their final component labels. For such processors, the output and postprocessing cycle consists of the following.

1. All such processors that are currently responsible for a black pixel of the image, connect their bus to all neighboring pixels also maintaining black pixels. The result is that there is a subbus over all figures with respect to the restriction of the image that remains in the SRM.
2. Exploiting the concurrent write capability, all processors in the last column of the SRM now broadcast the column (with respect to the entire image) label of their pixel. All processors receiving a value store this in C_R . Again, note that at output cycle c , this label is $n - c + 1$. All the processors that have not previously set the value of C_R receive this value and store it in C_R .
3. Unfortunately, multiple unique figures may now have the same component labels in terms of C_L and C_R . The labels are disambiguated as follows.
 - (a) Every processor in the last column that contains a black pixel connects its bus to its northern processor if and only if that processor maintains a black pixel. In a similar fashion, every processor in the last column that contains a black pixel connects its bus to its southern processor if and only if that processor maintains a black pixel. Next, perform two bus broadcast operations in the last column of the SRM over these subbusses so as to identify the topmost and bottommost pixels in every connected component over the restriction of the image to this last column.
 - (b) Each such processor in the last column of the SRM that represents a pixel in the image at position (i, j) prepares a record (C_L, C_R, i, j, T) . (Notice that all such records have identical values for j , which is included in the record for consistency.) Sort these, fewer than n , records using the integer packing result of [35]. After sorting, all records with identical C_L and C_R values will appear in contiguous positions in the last column ordered by their original position.

- (c) For each top/bottom pair (which are in adjacent processors in the last column of the SRM) representing identical labels, the bottom processor will broadcast a bit and observe whether or not this bit is received by its mate. This operation is somewhat intricate, involving several row and column broadcasts. This is done by first moving the labels back to their original unsorted location, and then using the standard technique of odd-even bus splitting, the bits are broadcast to find the topmost-bottommost pair of each group having identical C_L and C_R fields.
 - (d) Finally, move the labels back, and perform a bus-splitting-and-broadcast operation over the ordered set of data in the last column so as to broadcast a processor ID to be used as the T value in disambiguating labels with identical C_L and C_R fields.
4. In lockstep, shift the image to the right so as to output the next column of data (pixel and label information).

It should be noted that there is no static stage in this algorithm. However, the cost of the output and postprocessing stage, while asymptotically constant, requires a non-trivial amount of work and is primarily of theoretical interest.

Theorem 7 *Given an $n \times n$ digitized image, it can be labeled on the word-model systolic reconfigurable mesh of size n^2 using only a fixed number of broadcasts per input and preprocessing cycle, eliminating the static phase, and outputting the image in lockstep fashion with using $O(1)$ sorting.*

4. Conclusion

In this paper, we introduced a novel architecture, namely the *systolic reconfigurable mesh (SRM)*, which is motivated by combining the concepts of massively parallel reconfigurable architectures with that of attached systolic processors designed to be used in specialized domains. Efficient algorithms were developed that (i) process the data upon input and output, while avoiding any processing during a static state, (ii) consist of simple operations during each phase of processing, and (iii) have low constants and minimal total execution time. We presented algorithms for intermediate-level image processing tasks including histogramming, connectivity, convexity, and proximity. For labeling an $n \times n$ image, we presented and discussed three different algorithms. Open problems include a generic (one without a sorting routine and requiring $2n$ cycles only) labeling algorithm for the word-model and a convex hull algorithm that operates efficiently on an image containing multiple figures. Also, an interesting extension of these problems would be to investigate the situation where the image is larger than the mesh.

5. Acknowledgments

We would like to thank the referees for their valuable comments. Also thanks to Mr. Walter Yogi for his help in searching the literature.

- [1] H. M. Alnuweiri. Constant-time parallel algorithms for image labeling on a reconfigurable network of processors. *IEEE Trans. on Parallel and Distributed Systems*, pages 320–326, March 1994.
- [2] D Chin, J Pease, F Bernard, H. Taylor, and S Knight. The princeton engine: A real-time video system simulator. *IEEE Transactions on Consumer Electronics*, 34(2):285–297, 1988.
- [3] R. Cypher, J. L. C. Sanz, and L. Snyder. EREW PRAM and mesh connected computer algorithms for image component labeling. In *Proceedings of the IEEE Workshop on Computer Architecture for Pattern Analysis and Machine Intelligence*, pages 122–128, 1987.
- [4] M. J. B. Duff. CLIP4. *Special Computer Architectures for Pattern Processing*, pages 65–86, 1982.
- [5] Mary M. Eshaghian. Parallel algorithms for image processing on OMC. *IEEE Transactions on Computers*, 40(7):827–833, July 1991.
- [6] Mary M. Eshaghian, K. Kim, G. Nash, and D. B. Shu. Implementation and application of a gated connection network in image understanding architecture. In H. Li and Q. Stout, editors, *Reconfigurable Massively Parallel Computers*, pages 64–87. Prentice Hall, 1991.
- [7] Terry J. Fountain, K.N. Mathews, and Michael J. B. Duff. The CLIP7A image processor. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(3):310–319, 1988.
- [8] Dan W. Hammerstorm and Daniel P. Lulich. Image processing using one-dimensional processor arrays. *Proceedings of the IEEE*, 84(7):1005–1018, 1996.
- [9] M. C. Herbordt, C. C. Weems, and M. J. Scudder. Nonuniform region processing on SIMD arrays using the coterie network. *Machine Vision and Applications*, 5(2):105–125, 1992.
- [10] J. -W. Jang, H. Park, and V. K. Prasanna. A fast algorithm for computing histograms on a reconfigurable mesh. In *Proc. 4th Symposium on Frontiers of Massively Parallel Computation*, pages 244–251. IEEE, October 1992.
- [11] J. -F. Jenq and S. Sahni. Histogramming on a reconfigurable mesh computer. In *Proc. 6th International Parallel Processing Symposium*, pages 425–432. IEEE, 1992.
- [12] R. Miller, V. K. P. Kumar, D. I. Reisis, and Q. F. Stout. Parallel computations on reconfigurable meshes. *IEEE Trans. on Computers*, 42(6):678–692, June 1993.
- [13] R. Miller and Q. F. Stout. Parallel geometric algorithms for digitized pictures on mesh connected computers. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, March 1985.
- [14] R. Miller and Q. F. Stout. Efficient parallel convex hull algorithms. In *IEEE Transactions on Computers*, pages 1605–1618, December 1988.
- [15] V. K. Prasanna-Kumar and M. Mary Eshaghian. Parallel geometric algorithm for digitized pictures on mesh of trees. In *Proc. of IEEE International Conference on Parallel Processing*, pages 270–273, 1986.
- [16] Lorenz A. Schmitt and Stephen S. Wilson. The AIS-5000 parallel processor. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(3):320–330, 1988.
- [17] J. Rose, A. El Gamal, and A. Sangiovanni-Vincentelli. Architecture of field programmable gate arrays. *Proceedings of the IEEE*, July, 1993.
- [18] S. Hauck. The roles of FPGAs in programmable systems. *Proceedings of IEEE*, 86, 1998.
- [19] S. Schaumont, I. Verbauwhede, K. Kuetzer, and M. Sarrafzadeh. A quick safari through the reconfiguration jungle. *Proceedings of the 38th Design Automation Conference*, 2001.

- [20] Xilinx Inc. Xilinx platform FPGAs. *www.xilinx.com*.
- [21] K. Q. Li and V. Y. Pan. Parallel matrix multiplication on a linear array with reconfigurable pipelined bus system. *IEEE Transactions on Computers*, 50(2):519–525, 2001.
- [22] Mary M. Eshaghian and Lili Hai. An optically interconnected reconfigurable mesh. *Journal of Parallel and Distributed Computing*, 61:737–747, 2001.
- [23] Mary M. Eshaghian and Lili Hai. A glance at VLSI optical interconnects: From the modelings of the 1980s to today's MEMS implementations. *Handi-book on Innovative Computing*, 2004-2005.
- [24] K. Bondalapati. *Modeling and Mapping of Dynamically Reconfigurable Hybrid Architectures*. PhD thesis, University of Southern California, 2001.
- [25] K. Bondalapati and V. K. Prasanna. Reconfigurable computing: Architectures, models and algorithms. *Current Science*, 78(7):828–837, 2000.
- [26] Kiran Bondalapati and Viktor K. Prasanna. Reconfigurable computing systems. *Proceedings of IEEE*, July 2002.
- [27] R. Miller, V. K. P. Kumar, D. Reisis, and Q. F. Stout. Meshes with reconfigurable buses. *Proc. 15th MIT Conference on Advanced Research in VLSI*, pages 163–178, March 1988.
- [28] H. Li and M. Maresca. Polymorphic-torus network. *IEEE Trans. on Computers*, 38(9):1345–1351, September 1989.
- [29] C. C. Weems, S. P. Levitan, A. R. Hanson, and E. M. Riseman. The image understanding architecture. *International Journal of Computer Vision*, 2:251–182, 1989.
- [30] K. Nakano. A list of papers on algorithms for reconfigurable architectures. *Parallel Processing Letters*, 5(1):111–124, 1995.
- [31] A. Condon, R.E. Ladner, J. Lampe, and R. Sinha. Complexity of sub-bus mesh computations. Technical Report 93-10-02, Dept. of CS&EE, U. of Washington, 1993.
- [32] R. Miller and Q.F. Stout. *Parallel Algorithms for Regular Architectures: Meshes and Pyramids*. The MIT Press, Cambridge, Mass, 1996.
- [33] R.L. Graham. An efficient algorithm for determining the convex hull of a set of points in the plane. *Information Processing Letters*, 1:132–133, 1972.
- [34] B. F. Wang and G. H. Chen. Two-dimensional processor array with a reconfigurable bus system is at least as powerful as crw model. *Information Processing Letters*, 36:31–36, 1990.
- [35] S. Olariu, J.L. Schwing, and J. Zhang. Integer problems on reconfigurable meshes, with applications. *Journal of Computer and Software Engineering*, 1:33–45, 1993.