Parallel Computations on Reconfigurable Meshes

Russ Miller, *Member, IEEE,* V. K. Prasanna-Kumar, *Senior Member, IEEE,*
Dionisios I. Reisis, and Quentin F. Stout, *Senior Member, IEEE*

Abstract—**This paper introduces the *mesh with reconfigurable bus (reconfigurable mesh)* as a model of computation. The reconfigurable mesh captures salient features from a variety of sources, including the *CAAPP, CHiP, polymorphic-torus network,* and *bus automaton.* It consists of an array of processors interconnected by a reconfigurable bus system, which can be used to dynamically obtain various interconnection patterns between the processors. In this paper, we introduce a variety of fundamental data-movement operations for the reconfigurable mesh. Based on these operations, we also introduce new algorithms that are efficient for solving a variety of problems involving graphs and digitized images. The algorithms we present are asymptotically superior to those previously obtained for the aforementioned reconfigurable architectures, as well as to those previously obtained for the mesh, the mesh with multiple broadcasting, the mesh with multiple buses, the mesh-of-trees, and the pyramid computer, to name a few. Highlights include a logarithmic time algorithm to label the connected components of a graph given its adjacency matrix, as well as polylogarithmic time algorithms to solve problems involving convexity and connectivity of figures in images. We also show the power of reconfigurability by solving some problems, such as exclusive OR, more efficiently on the reconfigurable mesh than is possible on the PRAM.**

Index Terms— **Graph algorithms, image algorithms, mesh, mesh-of-trees, parallel algorithms, PRAM, pyramid computer, reconfigurable mesh, VLSI.**

I. Introduction

VLSI technology offers an environment for constructing parallel-processing systems that consist of thousands of processors. A very attractive interconnection scheme is the *two-dimensional mesh-connected computer (mesh)* because of its simplicity, regularity, and the fact that the interconnect wires occupy only a fixed fraction of the area no matter how large the mesh. However, since a *mesh of size* N is configured as an $N^{1/2} \times N^{1/2}$ grid of processors, the communication diameter (maximum of the minimum distance between any two processors in the network) is $\Theta(N^{1/2})$. Therefore, a lower bound on the time to solve nontrivial problems that involve combining data residing in processors far apart in a mesh of size N is $\Omega(N^{1/2})$. In order to achieve faster solutions to problems, researchers have studied related organizations that augment the mesh with additional communication links, while trying to keep the wire area small. Such organizations include the pyramid computer [1]–[3], the mesh-of-trees [4], [5], and meshes with broadcast buses [6]–[10].

For many algorithms, though, it is desirable that more than one interconnection scheme be present during their execution. In this paper, we introduce a model of computation that captures fundamental properties of *CHiP* [11], mesh computers augmented with broadcast buses [6]–[8], [12], the *bus automaton* [13], the *polymorphic-torus network* [14], and the *corterie network* in the latest version of the content addressable array parallel processor (CAAPP). We use the term *mesh with reconfigurable bus* or *reconfigurable mesh* to describe this model [46], [48]. The obvious advantage of working with a machine-independent abstract model is that it allows researchers to develop algorithms without worrying about certain technologically dependent parameters. In this paper, we introduce a number of fundamental data-movement operations that we then incorporate into algorithms to solve problems involving graphs and images. The algorithms we develop are asymptotically superior to previous algorithms for the aforementioned reconfigurable architectures. They also show that the reconfigurable mesh can provide more efficient solutions to problems than can other mesh-based architectures. In fact, we are able to give solutions to certain problems that are more efficient than those possible for the programmable random access memory (PRAM).

In Section II, we define the reconfigurable mesh. Section III illustrates the power of the reconfiguration scheme in basic operations on data as well as in sparse data movement by giving efficient implementations for fundamental data-movement operations such as random-access read/write, data reduction, and parallel prefix. These global operations form the foundation of algorithms appearing later in the paper and rely on the ability to reconfigure the bus in a variety of ways. Section IV presents embeddings of other parallel machines onto the reconfigurable mesh. It also presents algorithms that exploit the fundamental data-movement operations introduced in Section III to develop efficient solutions to several graph and image problems. Section V serves as the conclusion.

II. Mesh with Reconfigurable Bus

The *mesh with reconfigurable bus (reconfigurable mesh) of size* N consists of an $N^{1/2} \times N^{1/2}$ array of processors con-

Manuscript received May 31, 1988; revised June 7, 1991 and April 8, 1992.. The work of R. Miller was supported in part by the National Science Foundation (NSF) under Grant Nos. DCR-8 608 640 and IRI-8800514. The work of V. K. Prasanna-Kumar was supported in part by the NSF under Grant No. IRI-8710863. The work of D. I. Reisis was supported in part by DARPA under Contract No. F 33 615-84-K-1404 monitored by the Air Force Wright Aeronautical Laboratory. The work of Q. F. Stout was supported in part by the NSF under Grant No. DCR-8 507 851 and by an Incentives for Excellence award from Digital Equipment Corporation.

R. Miller is with the Department of Computer Science, State University of New York at Buffalo, Buffalo, New York 14260.

V. K. Prasanna-Kumar is with the Department of Electrical Engineering-Systems, EEB244, University of Southern California, Los Angeles, CA 90089-2562.

D. Reisis is with the Communication Laboratory, Division of Computer Science, Department of Computer Engineering, National Technical University of Athens, 15 773 Zographou, Athens, Greece.

Q. F. Stout is with the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, Michigan 48109-2122.

0018–9340/93$03.00 © 1993 IEEE

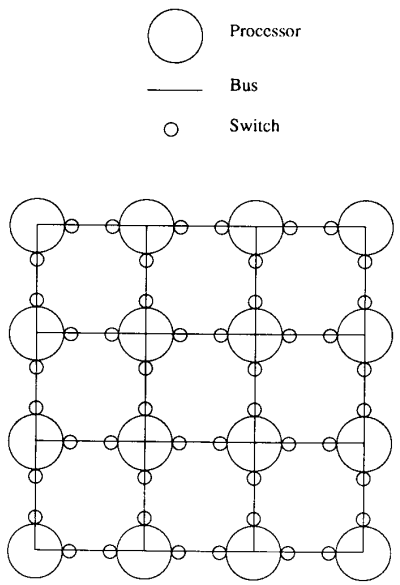


Fig. 1. Reconfigurable mesh.

nected to a grid-shaped reconfigurable broadcast bus, where each processor has four locally controllable bus *switches*, as shown in Fig. 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 is a maximal connected subset of the processors. Other than the buses and switches the reconfigurable mesh is similar to the standard mesh in that it has $\Theta(N)$ area in the word model of VLSI [15], under the assumption that processors, switches, and a link between adjacent switches occupy unit area.

We consider two models of bus arbitration in this paper. The *exclusive write model*, which mimics the exclusive write capability of the exclusive write PRAM (EW PRAM), allows only one processor to broadcast at any given time to a subbus shared by multiple processors. We assume that the value broadcast consists of $O(\log N)$[1] bits. The *common write model*, which mimics the common write version of the concurrent write PRAM (CW PRAM), allows multiple processors to simultaneously broadcast to the same subbus so long as they all broadcast the same value and it is only a single bit. The focus of this paper is on the exclusive write reconfigurable mesh. In fact, all algorithms presented in this paper are for the exclusive write model except for the component labeling algorithm associated with Theorem 1, which uses the common write model.

Two computational models will be discussed in this paper with respect to the delay that a broadcast requires. The *unit-time delay model* will assume that all broadcasts take $\Theta(1)$ time, as is the assumption in [6]–[9], [16] for models that assume various broadcasting strategies. We will also consider the *log-time delay model* in which it is assumed that each broadcast takes $\Theta(\log s)$ time to reach all the processors connected to its subbus, where s is the maximum number of switches in a minimum switch path between two processors connected on the bus. Obviously, any algorithm taking T time steps on the unit-time delay model can be completed in $O(T \log N)$ time steps on the log-time delay model. In some circumstances we are able to develop algorithms for the log-time delay model that are less than a factor of $\log N$ slower than the algorithms for the unit-time delay model. An example of this occurs in Proposition 4.

Many of the algorithms introduced in this paper will continually reconfigure the system by setting the switches to give the desired substructures. A number of algorithms have previously been derived that exploit row and column broadcasts [6], [8], [9], [16]. Many of these algorithms use row (column) broadcasts simultaneously within every row (column). This technique can be exploited on the reconfigurable mesh by having each processor set its switches to disconnect its column (row) links, creating $N^{1/2}$ separate row (column) buses. See Fig. 2. That is, processor $P_{i,j}$ has access to the broadcast buses of row i and column j, though complete row and column buses cannot be used simultaneously. Notice that by setting the switches properly, sub-row (column) buses can be created within each row (column), sub-reconfigurable meshes can be created, a global broadcast bus can be created, a bus can be created within sets of contiguously labeled processors, and so forth.

For some of the algorithms presented in this paper, a row-major ordering of the processors will be used, where processor $P_{i,j}$, $0 \leq i, j \leq N^{1/2} - 1$ has row-major index $iN^{1/2} + j$. For other algorithms, it is useful to view the mesh as a linear array of processors, by using, for example, a snake-like indexing of the processors (Fig. 3). Define the *predecessor* of each processor to be the processor with next lowest index, and the *successor* of each processor to be the processor with next highest index.

III. Data Movement Using Reconfigurable Buses

Data-movement operations form the foundation of numerous parallel algorithms for machines constructed as an interconnection of processors. In fact, algorithms designed in terms of fundamental *abstract data-movement operations* provide the possibility of portability to architecturally related machines. This notion can be viewed as the parallel analog to designing serial algorithms in terms of *abstract data types*. Therefore, the algorithms given in this paper will be described in terms of such operations. In this section, a variety of these fundamental operations are given for the reconfigurable mesh.

We first introduce a technique called *bus splitting*, which shows how the processors can exploit the ability to locally control the effective size of subbuses. Suppose we want to compute the logical OR of $N^{1/2}$ bits of data stored one bit per processor in the ith row of the reconfigurable mesh, storing the result in processor $P_{i,0}$, for all $0 \leq i \leq N^{1/2} - 1$. This can be accomplished as follows.

Set the switches so that each row is connected by a disjoint subbus. Next, each processor $P_{i,j}$ that has a "1" as its data value *splits* its bus by setting its eastern switch to disconnect its row bus. Then, each processor $P_{i,j}$ that has a "1" as

[1] All logarithms in this paper are to base 2, unless specified otherwise.

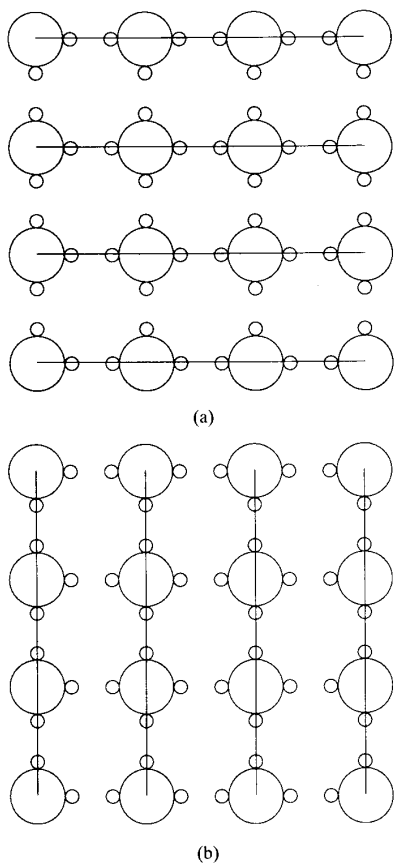


Fig. 2. Switches set for row and column broadcasts. (a) Switch settings to create an independent broadcast bus within every row. (b) Switch settings to create an independent broadcast bus within every column.

0	1	2	3
7	6	5	4
8	9	10	11
15	14	13	12

Fig. 3. Snake-like indexing of a mesh.

its data value broadcasts the "1" on its subbus. Processor $P_{i,0}$, for all $0 \leq i \leq N^{1/2} - 1$, will receive on its row subbus the westernmost "1" in its row, if such a value exists. Notice that if it is desired to find the logical OR of the data stored in all processors, then first the OR of each row can be determined followed by the OR of these values in the first column. Alternatively, the logical OR of the data in all processors can be determined by setting the switches so that all processors are connected by a single linear bus (following the snake-like indexing), and then using a single bus-splitting step, where each processor that contains a "1" splits the bus by disconnecting the switch between itself and its successor. A broadcast then informs processor $P_{0,0}$ as to the logical OR of the values.

Proposition 1: Given a reconfigurable mesh of size N, in which each processor stores a bit of data, the logical OR of the data in each row (column), or in the entire reconfigurable mesh, can be determined in $\Theta(1)$ time using the unit-time delay model, and in $\Theta(\log N)$ time using the log-time delay model. □

The reconfigurable-bus scheme can be superior to other parallel models for some computations. Consider, for example, computing the exclusive OR (EXOR) of $N^{1/2}$ values stored in a row of the mesh. It should be noted that in [17] it has been shown that the EXOR cannot be computed in $O(1)$ time on a PRAM using a polynomial number of processors. The ability to reconfigure the bus of the reconfigurable mesh, however, allows us to compute EXOR function in $\Theta(1)$ time using the unit-time delay model and in $\Theta(\log N)$ time using the log-time delay model.

Proposition 2: Given a reconfigurable mesh of size N, the EXOR of $N^{1/2}$ bits of data, initially stored one bit per processor in a row (column), can be computed in $\Theta(1)$ time using the unit-time delay model, and in $\Theta(\log N)$ time using the log-time delay model.

Proof: Without loss of generality, assume that the $N^{1/2}$ values are stored one per processor in row 0. Let d_j denote the bit in processor $P_{0,j}$. The computation of the EXOR proceeds by first computing the EXOR of the values stored in even-numbered processors, then computing the EXOR of the data in odd-numbered processors and finally combining these two results.

Consider the EXOR-computation of the data stored in $P_{0,2k}$, $0 \leq k \leq (N^{1/2}/2) - 1$. Two steps are involved in this task, namely configuring the bus and then computing the EXOR as a combination of broadcast and arithmetic operations. The processors in the $(2k)$th and $(2k + 1)$th columns configure the bus based on the data in $P_{0,2k}$, $0 \leq k \leq (N^{1/2}/2) - 1$, as follows:

1) Configure the bus so that processors in column $2k$ and processors in column $2k + 1$, $0 \leq k \leq (N^{1/2}/2) - 1$, share a subbus. Every processor $P_{0,2k}$, $0 \leq k \leq (N^{1/2}/2) - 1$, broadcasts its data value on its subbus, after which every PE $P_{i,j}$ knows the value stored in PE $P_{0,2\lfloor j/2 \rfloor}$.
2) If the value in $P_{0,2k}$ is a 0, then all the even-numbered processors in columns $2k$ and $2k + 1$ split the bus above and below them: $P_{2i,2k}$, $0 \leq i \leq (N^{1/2}/2) - 1$, sets to OFF the switch between itself and $P_{2i+1,2k}$ $(i \neq (N^{1/2}/2) - 1)$ and also the switch between itself and $P_{2i-1,2k}$ $(i \neq 0)$. Also, $P_{2i,2k+1}$ sets to OFF the switches between itself and the processors $P_{2i+1,2k+1}$ $(i \neq (N^{1/2}/2) - 1)$ and $P_{2i-1,2k+1}$ $(i \neq 0)$ (Fig. 4).
3) If the value in $P_{0,2k}$ is a "1" then $P_{2i,2k}$ splits the bus between itself and $P_{2i,2k+1}$. $P_{2i+1,2k}$ splits the bus between itself and $P_{2(i+1),2k}$ $(i \neq (N^{1/2}/2) - 1)$. $P_{2i+1,2k+1}$ splits the bus between itself and $P_{2i+1,2(k+1)}$ $(k \neq (N^{1/2}/2) - 1)$. Also, the switch between $P_{2i,2k+1}$ and $P_{2i+1,2k+1}$ $(i, k \neq (N^{1/2}/2) - 1)$ is set to OFF (Fig. 4).

First the mesh is configured as in step 1 above. Then, one

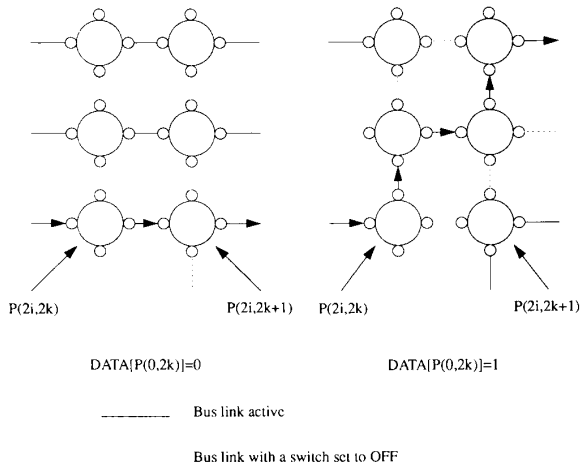


Fig. 4. Switch setting for the EXOR computation.

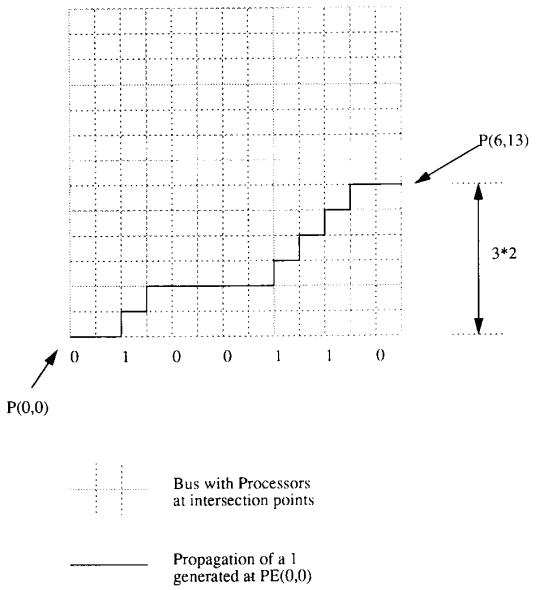


Fig. 5. EXOR computation of data in even-numbered processors on a 14 x 14 reconfigurable mesh.

of steps 2 or 3 is followed in each pair of columns. Notice that in case of a "0" in $P_{0,2k}$, the broadcast data on the bus reaching $P_{2i,2k}$ will also reach $P_{2i,2(k+1)}$. In case of a "1" in $P_{0,2k}$, a data-item broadcast on the bus follows the route from $P_{2i,2k}$ to $P_{2(i+1),2(k+1)}$.

Assume now that the bus is configured as above. A "1" is generated and broadcast from processor $P_{0,0}$. Due to the above configuration of the bus, the broadcast "1" will be read by (and pass through) exactly one processor $P_{2l,2j}$ in each column $2j$, where $l = \sum_{k=0}^{j-1} d_{2k}$, $1 \leq j \leq (N^{1/2}/2)$. Thus, if processor $P_{2i,N^{1/2}-1}$ is the processor in column $N^{1/2}-1$ that reads a "1" from the bus, then the sum of the 1's in even-numbered processors of row 0 is i. This processor can broadcast i to all the processors in the mesh. The EXOR computation of data in even-numbered processors on a 14×14 reconfigurable mesh is shown in Fig. 5. Notice that in this example the number of "1"s is 3 and $2i$ is 6. In a similar fashion, the sum of the 1's in the odd-numbered processors (in row 0) may be computed. Finally, from these two values, the EXOR of $N^{1/2}$ bits of data, initially stored in the same row (column), may be computed in unit time. Therefore, the running time is as claimed. □

Note that in each column, the row index of the PE receiving a 1 is a measure of the number of 1's up to that column in row 0. Thus, the previous result can be extended in a natural fashion to the following.

Corollary 1: Suppose on a reconfigurable mesh of size N, each processor in row (column) j stores a bit of data d_j, $0 \leq j \leq N^{1/2}-1$. The prefix computation of $\mathcal{F}_j$, where $\mathcal{F}_j = \sum_{i=0}^{j} d_i$ is to be stored in row j, $0 \leq j \leq N^{1/2}-1$, can be performed in $\Theta(1)$ time using the unit-time delay model and in $\Theta(\log N)$ time using the log-time delay model. □

Many algorithms are designed to reduce data at intermediate stages of the algorithm. It is, therefore, often useful to be able to efficiently perform fundamental operations on reduced sets of data. The maximum of $N^{1/2}$ values $x_0, \cdots, x_{N^{1/2}-1}$ initially stored in a row of a reconfigurable mesh of size N can be determined as follows. First, column broadcasts can be used so that every processor $P_{i,j}$ contains entry x_j. Next, within every row i, processor $P_{i,i}$ uses a row broadcast to inform all processors $P_{i,j}$ of the value of x_i, $0 \leq i,j \leq N^{1/2}-1$. At this point, every processor $P_{i,j}$ contains data values x_i and x_j. Next, every processor computes the Boolean result of "$x_j < x_i$." In every column j, the logical OR of these results (using the algorithm associated with Proposition 1 to store the result in $P_{0,j}$) can be used to decide whether or not x_j is a maximum. Notice that a result of "0" in $P_{0,j}$ indicates that x_j is a maximum, while a "1" indicates that it is not. Since there may be more than one processor in row 0 storing the maximum value, bus splitting on row 0 is used to inform $P_{0,0}$ as to the maximum value, which can then be broadcast to all processors. Notice that the minimum can be computed similarly.

Proposition 3: Given a reconfigurable mesh of size N, in which no more than one processor in each column stores a data value, the maximum (minimum) of these $O(N^{1/2})$ data items can be determined in $\Theta(1)$ time using the unit-time delay model, and in $\Theta(\log N)$ time using the log-time delay model. □

By a somewhat more complicated sequence, Valiant's PRAM algorithm for finding the maximum [18] can be simulated on a reconfigurable mesh to find the maximum of N values, assuming they are stored one value per processor. Assume that the mesh is divided into $N^{1/2}$ disjoint blocks of size $N^{1/4} \times N^{1/4}$. Suppose the maximum of the data in each block has been determined. Then, we need to determine the maximum of $N^{1/2}$ values using N processors in the array. After moving the data to appropriate locations, this can be done in $O(1)$ time using Proposition 3 in the unit-time delay model. Recursively applying this idea leads to an algorithm with running time that obeys the recurrence $T(N) = T(N^{1/2}) + \Theta(1)$ in the unit-time delay model and $T(N) = T(N^{1/2}) + \Theta(\log N^{1/2})$ in the log-time delay model.

Proposition 4: Given a set of data items S of size N stored one per processor on a reconfigurable mesh of size N, the maximum (minimum) value of S can be determined in $\Theta(\log\log N)$ time using the unit-time delay model and in $\Theta(\log N)$ time using the log-time delay model. □

Combining the above approach with Proposition 2 leads to:

Corollary 2: Given a set of bits S of size N stored one per processor on a reconfigurable mesh of size N, the EXOR of all items in S can be computed in $\Theta(\log\log N)$ time using the unit-time delay model and in $\Theta(\log N)$ time using the log-time delay model. □

Parallel prefix is an important operation that can be used to sum values, broadcast data, solve problems in image processing, solve graph problems, and so forth [19]. Assume processor P_i, $0 \leq i \leq N-1$, initially contains data element a_i. The *parallel prefix problem* requires every processor P_i, $0 \leq i \leq N-1$, to determine the ith initial prefix $a_0 \otimes a_1 \otimes \cdots \otimes a_i$, where $\otimes$ is a binary associative operator.

An efficient mesh-based algorithm is straightforward. Assume that the indexing of the processors is row-major. First, perform parallel prefix within each row so that each processor knows the initial prefix of those values restricted to its row. Next, in the last column perform parallel prefix to determine row-wise prefix solutions. Finally, within each row, broadcast the prefix of the previous rows so that all processors can update their entry appropriately.

Notice that parallel prefix can be simultaneously computed in every row (or column) of the reconfigurable mesh in $\log N^{1/2}$ iterations by appropriately setting switches, broadcasting, and updating values at each iteration. During the ith iteration of the row version, switches are set so that every row is grouped into disjoint linear *strings* consisting of 2^i processors, $1 \leq i \leq \log N^{1/2}$. After configuring the switches, within each string processor $P_{2^{i-1}-1}$ (i.e., the easternmost processor of the low-order substring of size 2^{i-1}) broadcasts its prefix value to all processors in its string. All processors $P_{2^{i-1}} \cdots P_{2^i-1}$ update their prefix value by applying $\otimes$ to the broadcast value and their current value.

Finally, it should be noted that if the snake-like indexing scheme is used, then the problem only requires a single $\log N$ iteration prefix operation to be performed within the string of size N.

Lemma 1: Given a set $S = \{a_i\}$ of N values, distributed one per processor on a reconfigurable mesh of size N so that processor P_i contains a_i, $0 \leq i \leq N-1$, and a unit-time binary associative operation $\otimes$, in $\Theta(\log N)$ time using the unit-time delay model and in $\Theta(\log^2 N)$ time using the log-time delay model, the parallel prefix problem can be solved so that each processor P_i knows $a_0 \otimes a_1 \otimes \cdots \otimes a_i$. □

It is often desirable to model PRAM algorithms on other machines. The CRCW PRAM consists of a set of processors and a shared memory in which concurrent reads from the same memory location are allowed as are concurrent writes to the same location. In the case of concurrent writes, a predefined scheme is used to decide which value succeeds. In order to efficiently simulate the CRCW PRAM, one must be able to efficiently simulate the concurrent read and concurrent write properties. Define random access read (RAR) to be a data-movement operation that models a concurrent read, in which each processor knows the index of another processor from which it wants to read data [20]. Similarly, a random access write (RAW) will model a concurrent write in that each processor knows the index of a processor that it wishes to write to [20]. In case of multiple writes to the same processor, a tie-breaking scheme is used, such as minimum or maximum data value, or arbitrarily letting one value succeed. Such data-movement operations on SIMD computers using N processors can be performed in the same time as it takes to sort N numbers. However, on the reconfigurable mesh, we give algorithms whose running times depend on the number of data items to be moved.

The algorithms we give in this section are actually concerned with more general versions of these operations. The operations that we consider have PE's attempting to read or write information based on keys, where the key may or may not be the index of a processor. In order to maintain consistency during concurrent read and concurrent write operations, it will be assumed that there is at most one *master record*, stored in some processor, associated with each unique key. In a concurrent read, each PE generates a fixed number of *request records*, where each request record specifies a key that the PE wishes to receive information about. In a concurrent write, each PE generates a fixed number of *update records*, where each update record specifies the key and data field corresponding to the key that the PE wishes to update. It should be noted that for many applications, a PE will maintain master records and also generate request or update records.

Lemma 2: Given a reconfigurable mesh of size N, in $O(k^{1/2})$ time using the unit-time delay model, and in $O(k^{1/2}\log N)$ time using the log-time delay model, k data items may be moved in a RAR or RAW, where $k \leq N$.

Proof: The algorithm given in this proof can be applied once to perform a RAW and twice to perform a RAR. The basic data movement is accomplished by moving the data (update or request records) into the northwest block of size $\Theta(k)$, making multiple copies of this block throughout the mesh, and then using mesh-computer RAR's/RAW's within each block to either update or obtain information regarding the appropriate master records. The details of moving the k data items into the northwest submesh of size $\Theta(k)$ follow.

1) Each processor that has a data item (update record in the case of RAW, request record in the case of RAR) to be moved marks itself *active*.
2) Using row subbuses, each row determines the westernmost active data item (if any) in the row. Call each such item *live*.
3) Using Corollary 1, where rows with live items use a data value of "1" and other rows use "0," each row with a live item determines the item's *rank* with respect to the live items, i.e., the number of live items in preceding rows. Let L_1 denote the total number of live items.
4) For $j = 1$ to $\lceil L_1/\lceil k^{1/2}\rceil\rceil$, using the row buses, the items with rank $(j-1)\lceil k^{1/2}\rceil$ through $j\lceil k^{1/2}\rceil - 1$ are moved to the columns 0 through $\lceil k^{1/2}\rceil - 1$, respectively. Then, using column buses, they are moved to row $j-1$. When this loop is completed, the live items have been moved to

the northwest corner, occupying positions $0 \cdots \lceil k^{1/2} \rceil - 1$ of rows $0 \cdots \lfloor L_1/\lceil k^{1/2} \rceil \rfloor - 1$, and positions $0 \cdots L_1 - \lceil k^{1/2} \rceil \lfloor L_1/\lceil k^{1/2} \rceil \rfloor - 1$ of row $\lceil L_1/\lceil k^{1/2} \rceil \rceil - 1$. These live items are now marked inactive.

5) Repeat steps 2 and 3 using columns instead of rows, and northernmost instead of westernmost, where L_2 is used to denote the number of live items. Then, using a process similar to step 4, move these live items to the northwest corner, starting in row $\lceil L_1/\lceil k^{1/2} \rceil \rceil$ and ending at row $\lceil L_1/\lceil k^{1/2} \rceil \rceil + \lceil L_2/\lceil k^{1/2} \rceil \rceil - 1$.
6) Steps 2 through 5 are repeated, alternating between using rows and using columns to select live items, where the live items of each iteration are placed in the northwest corner in rows following the previous iteration, until all k items have been moved to the northwest corner. One can show that live items need to be selected, at most, $2\lceil k^{1/2} \rceil - 1$ times, and that the total number of rows used in the corner is at most $3\lceil k^{1/2} \rceil - 1$. Now, using standard mesh operations, the items can be compressed into $k/\lceil k^{1/2} \rceil$ rows of width $\lceil k^{1/2} \rceil$.

In the unit-delay model, steps 2 and 3 take constant time, step 4 takes $O(L/\lceil k^{1/2} \rceil)$ time per iteration, and step 6 takes $O(\lceil k^{1/2} \rceil)$ time. Therefore, the total time is $O(k^{1/2})$ time in the unit-time delay model and in $O(k^{1/2} \log N)$ time in the log-time delay model.

Note that if k is not known in advance, then one could perform the above algorithm, omitting step 4 and the final compression, to determine k in $O(k^{1/2})$ time in the unit-time delay model, and in $O(k^{1/2} \log N)$ time in the log-time delay model. By different techniques k can be determined in $\Theta(\log k)$ time in the unit-time delay model and $\Theta(\log k \log N)$ time in the log-time delay model, but this faster determination would not improve the total time to perform the RAR or RAW.

The second phase of the algorithm consists of moving the records in the northwest submesh of size $\Theta(k)$ to their final destinations. Perform $\Theta(k^{1/2})$ row broadcasts, followed by $\Theta(k^{1/2})$ column broadcasts to create $\Theta(N/k)$ disjoint copies of the northwest submesh of size $\Theta(k)$. Now, within each submesh, perform sort-based mesh operations to complete the operation.

The total time for the second phase is $\Theta(k^{1/2})$ in the unit-time delay model and $\Theta(k^{1/2} \log N)$ time in the log-time delay model. Hence, the running time of the RAR and RAW is as claimed. □

For the restricted RAR and RAW operations (in which reads/writes are via predetermined processor locations), more efficient algorithms can be used if the distribution of *source processors*, i.e., those processors sending data, as well as *destination processors*, i.e., those processors receiving data, is uniform over the mesh.

Proposition 5: If the maximum number of source and destination processors within any block of size k^2 is k, $1 \leq k \leq N^{1/2}$, then RAR and RAW can be performed in $\Theta(\log N)$ time under the unit-time delay model and in $\Theta(\log^2 N)$ time under the log-time delay model.

Proof: Assume that the number of records to be moved is n_r, which can be at most $N^{1/2}$. The general idea is to first move this data to the diagonal processors of the reconfigurable mesh and then distribute them to their destination.

To move the records to the diagonal processors of the mesh, we use an iterative procedure that merges four blocks of size k^2 to obtain a block of size $4k^2$, where $2 \leq k \leq N^{1/2}/2$. Suppose there are r_i records in each block of size k^2, $r_i \leq k$, $1 \leq i \leq 4$, where r_1 represents the number of records in the northwest block, r_2 represents the number of records in the northeast block, r_3 represents the number of records in the southwest block, and r_4 represents the number of records in the southeast block. Assume that these records are stored in diagonal processors $P_{0,0}$ through P_{r_i-1,r_i-1}, where the indices are defined locally within blocks of size k^2. Using a fixed number of row and column broadcasts, all processors in the block of size $4k^2$ know the values of r_i, $1 \leq i \leq 4$. Another pair of row and column broadcasts for the northeast, southwest, and southeast blocks will move the n_2 records from the northeast block to diagonal processors P_{n_1,n_1} through $P_{n_1+n_2-1,n_1+n_2-1}$, the n_3 records from the southwest block move to diagonal processors $P_{n_1+n_2,n_1+n_2}$ through $P_{n_1+n_2+n_3-1,n_1+n_2+n_3-1}$, and the n_3 records from the southwest block move to diagonal processors $P_{n_1+n_2+n_3,n_1+n_2+n_3}$ through $P_{n_1+n_2+n_3+n_4-1,n_1+n_2+n_3+n_4-1}$, with respect to the block of size $4k^2$.

After the four blocks of size $N/4$ are merged into a single block of size N, all n_r active records will be stored in diagonal processors $P_{0,0}$ through P_{n_r-1,n_r-1}. These data are sorted using standard techniques based on rank computation.

Now, using a top-down recursive solution strategy, move the contiguous set of diagonal elements from the diagonal of the reconfigurable mesh of size N to the diagonal of the proper reconfigurable mesh of size $N/4$. This is accomplished by a fixed number of row and column broadcasts. Continue in this fashion until all records have reached their destination.

Row and column broadcasts take $\Theta(1)$ time in the unit-time delay model and $\Theta(\log N)$ time in the log-time delay model. Sorting data that initially resides one element per diagonal processor takes $\Theta(\log N)$ time in the unit-time delay model and $\Theta(\log^2 N)$ time in the log-time delay model. Notice, however, that sorting is only done once in the algorithm, after the bottom-up stage is complete and the data are along the main diagonal, and before the top-down distribution stage is initiated. The running time for the top-down and bottom-up stages of the algorithm under the unit-time delay model is given by $T(N) = T(N/4) + \Theta(1)$, which is $\Theta(\log N)$, while the running time for these two stages of the algorithm under the log-time delay model is given by $T(N) = T(N/4) + \Theta(\log N)$, which is $\Theta(\log^2 N)$. Therefore, a single sort step does not affect the asymptotic running time of the algorithm, and the running times are as claimed. □

The next operation we consider is data reduction. Assume that each processor has at most one record having a *key* field and a *data* field. *Data reduction* will perform an associative binary operation on the data of records having the same key. At the end of the data-reduction operation, each processor with key k will have the result of the operation performed over all data items with key k.

Proposition 6: Given an associative binary operator $\otimes$, data reduction can be performed on k distinct keys in $\Theta(k^{1/2}+$

$\log N)$ time on a reconfigurable mesh of size N under the unit-time delay model and in $\Theta(k^{1/2}\log N + \log^2 N)$ time under the log-time delay model. At the end of the operation each processor knows the result of applying $\otimes$ over all data items with its key.

Proof: It will be shown later that the number of distinct keys can be computed in $O(k^{1/2}+\log N)$ time under the unit-time delay model and in $O(k^{1/2}\log N + \log^2 N)$ under the log-time delay model. Further, once k is known, the keys can be mapped to values $0, \cdots, k-1$ in the same time. Therefore, assume that the number of keys, k, is known, and that the keys are labeled $0, 1, \cdots, k-1$. The basic idea of the algorithm is similar to the merge step of the algorithm presented in Proposition 5. Initially, data reduction is performed in $O(k^{1/2})$ time using a sort-based mesh computer algorithm within blocks of size k. Once this is complete, blocks are merged iteratively while continuing to reduce data, until the data are reduced over the entire mesh.

Details of the algorithm follow.

1) Partition the reconfigurable mesh of size N into disjoint blocks of size k. In each such block sort the records according to their key and perform $\otimes$ over the set of data associated with each key. Next, create one entry to represent each key, including keys that do not appear in the block. For a key that appears in the block, this entry will contain the result of applying $\otimes$ over all data values within the block associated with the key. For a key that does not appear in the block, the data value of the entry will be *nil*. Mesh algorithms will complete this task in $\Theta(k^{1/2})$ time.
2) Within each block of size k, move the record with key i to processor P_i, where the indexing scheme of the processors is *column-major order*.
3) Merge four blocks of size b^2 into a block of size $4b^2$ as follows. The k records are stored in the first $\lceil k/b \rceil$ columns of each block of size b^2. There is exactly one record in each block of size b^2 corresponding to each of the k keys. Since records are sorted by keys, records with the same key are located in the corresponding processors of each block of size b^2. Using row and column broadcasts, combine the corresponding entries so that $\otimes$ is performed over the four corresponding values, with the results being stored in the corresponding PE's of the northwest block of size b^2. Since there are $\Theta(k/b)$ columns, this operation requires $\Theta(k/b)$ time.
4) Distribute the records such that they are in column-major order in columns 0 to $(\lceil k/2b \rceil - 1)$ of the block of size $4b^2$. This can be done, as in the previous step, in $O(k/b)$ time by exploiting the diagonal processors.
5) Starting with $b = k^{1/2}$, repeat steps 3 and 4 until $b = N^{1/2}$.
6) Using the algorithm associated with Proposition 2, move all k records into the northwest submesh of size k.
7) Copy the data in this northwest submesh to each submesh of size k using row and column broadcasts.
8) Within each submesh of size k, the data corresponding to each key can be moved to the processors with that key in $O(k^{1/2})$ time by using mesh computer algorithms.

The initial setup takes $O(k^{1/2})$ time, the ith merging operation uses $O(k^{1/2}/2^i + 1)$ broadcasts, and the final routing step can be completed in $O(k^{1/2})$ time. Therefore, the running time of the algorithm is as claimed. □

As previously mentioned, the number of distinct keys can also be computed in $O(k^{1/2}+\log N)$ time under the unit-time delay model and in $O(k^{1/2}\log N + \log^2 N)$ time under the log-time delay model. This can be accomplished as follows. Assume that there exists $\log^2 N$ or fewer distinct keys. Use the key-reduction algorithm (trivially modified) followed by a logical OR to decide whether or not every key was represented among the $\log^2 N$ finalists. If every key was not represented, then assume the number of keys is a multiplicative factor of 2 greater than the previous assumption. Continue this approach until all keys are represented in the final set. The algorithm is similar to the previous one, motivated from a key counting algorithm in [3] and is therefore omitted.

Corollary 3: Given one keyed item per processor on a reconfigurable mesh of size N, the number of distinct keys can be determined in $\Theta(k^{1/2} + \log N)$ time under the unit-time delay model and in $\Theta(k^{1/2}\log N + \log^2 N)$ time under the log-time delay model. □

IV. Applications

In this section, we illustrate the power of the reconfigurable mesh by giving efficient parallel algorithms to solve graph and image problems. Many of the solutions rely on the fundamental data-movement operations given in the previous section. We also show that the reconfigurable mesh can exploit its numerous communication patterns to efficiently simulate other low wire-area organizations. Section IV-A shows that the reconfigurable mesh can efficiently simulate the mesh-of-trees and pyramid, two architectures for which numerous efficient algorithms already exist. In Section IV-B, efficient algorithms are given for solving a variety of graph problems, while in Section IV-C efficient algorithms are given for solving problems in image analysis. It should be noted that the algorithms given in Sections IV-B and IV-C are more efficient than those that can be obtained by a direct simulation of mesh-of-trees or pyramid algorithms.

A. Simulations

Let $\mathcal{G}$ be a family of undirected graphs $G^n = (V^n, E^n)$, with $|V^n| = n$, where the vertices represent processors and the edges represent communication links between the processors of a parallel computer. A $(p(n), c(n))$-*embedding* of $\mathcal{G}$ into reconfigurable meshes consists of the following. For every n there is a map ρ^n of V^n into the processors of a reconfigurable mesh M^n of size n, a partition of E^n into subsets $E_0^n, \cdots, E_{c(n)}^n$, and a collection of switch settings $\pi_1^n, \cdots, \pi_{c(n)}^n$ for M^n, such that the following hold.

1) No PE of M^n has more than $p(n)$ vertices mapped onto it via ρ^n.
2) E_0^n consists of those edges $(u, v) \in E^n$ such that the endpoints are mapped to the same processor of M^n, i.e., such that $\rho^n(u) = \rho^n(v)$.

3) For each i, $1 \leq i \leq c(n)$, the subbuses generated by π_i^n are in 1–1 correspondence with the edges of E_i^n, where if (u, v) is an edge of E_i^n, then π_i^n has created a subbus that is a path with one endpoint at $\rho^n(u)$ and the other endpoint at $\rho^n(v)$.
4) In $O(p(n))$ time, each processor of M^n can determine the vertices of V^n mapped onto it.
5) For every i, $1 \leq i \leq c(n)$, in $\Theta(1)$ time each processor of M^n can determine its switch settings given by π_i^n and, if it is an endpoint of a subbus, it can determine which simulated processor corresponds to that end.

Notice that, by the definition, a vertex may appear as an endpoint of at most one edge in any given partition of E^n. One significant point of this definition is that a $(p(n), c(n))$-embedding does not correspond to a standard layout in which $c(n)$ communication layers are used. There are two primary differences between such a layout and our embedding. The first is that in a layout there may be vias connecting the layers, so that a single communication edge may be on multiple layers, while the $(p(n), c(n))$-embedding does not allow this. The second is that a layout allows the edges to form an arbitrary planar grid-graph, while in the $(p(n), c(n))$-embedding each processor (vertex) can be connected to at most one subbus (and hence at most one edge) at a time. For example, a planar grid of size N in which each processor is connected to its four nearest neighbors is normally viewed as a one-level layout, but a reconfigurable mesh needs four sets of switch settings to simulate all the edges of the grid.

We assume that a single time step of an algorithm for $\mathcal{G}$ consists of a fixed amount of local calculation, followed by a communication step in which each processor is involved with sending or receiving at most one message, consisting of a single word.

Given a $(p(n), c(n))$-embedding of $\mathcal{G}$, we say that an algorithm for $\mathcal{G}$ is *normalized* if there is a constant k, such that for each n and each communication step of the algorithm there is a set $\{i_1, i_2, \cdots, i_k\}$ such that k communication steps are involved on the reconfigurable mesh, one each restricted to edges in $E_{i_1}, E_{i_2}, \cdots, E_{i_k}$, and $\{i_1, i_2, \cdots, i_k\}$ can be determined in $\Theta(1)$ time.

Proposition 7: Given a family of graphs $\mathcal{G}$ representing a parallel architecture, given a $(p(n), c(n))$-embedding of $\mathcal{G}$ into reconfigurable meshes, and given an algorithm $\mathcal{A}$ for $\mathcal{G}$ taking $T(N)$ time on G^N, then on the reconfigurable mesh of size N

1) $\mathcal{A}$ can be simulated in $O((p(N) + c(N))T(N))$ time on the unit-time delay model and in $O((p(N) + c(N))T(N) \log N)$ time on the log-time delay model and
2) if $\mathcal{A}$ is normalized, then it can be simulated in $O(p(N)T(N))$ time on the unit-time delay model and in $O(p(N)T(N) \log N)$ time on the log-time delay model.

Proof: The simulation is quite staightforward. In each simulation step, each reconfigurable mesh processor first performs all the local calculations of the $p(N)$ or fewer processors mapped onto it and then simulates all communication where both ends are processors it is simulating (i.e., it simulates those communication links in E_0^N with endpoints in it). If the algorithm is not normalized, then at most $c(N)$ interprocessor communication operations are now performed. In the ith operation, $1 \leq i \leq c(N)$, each processor sets its switches according to π_i^N, determines if it is an endpoint of a subbus, and if it is an endpoint then sends a message or receives it (or neither, if the communication link is not used in this time step) for the simulated processor corresponding to its end of the communication link. If the algorithm is normalized, then only a fixed number of interprocessor communication operations are performed. In the unit-time delay model the initial local calculations take $O(p(N))$ time, and each interprocessor communication operation takes $\Theta(1)$ time, giving the time claimed. □

We will apply this theorem to two specific architectures of particular interest, namely the mesh-of-trees and the pyramid.

A *mesh-of-trees of base size* N, where N is an integral power of 4, has a total of $3N - 2N^{1/2}$ processors. N of these are base processors arranged as a mesh of size N. Above each row and above each column of the mesh is a perfect binary tree of processors. Each row (column) tree has as its leaves an entire row (column) of base processors. All row trees are disjoint, as are all column trees. Every row tree has exactly one leaf processor in common with each column tree. Each base processor is connected to six other processors (assuming they exist): four neighbors in the base, a parent in its row tree, and a parent in its column tree. Each processor in a row or column tree that is neither a leaf nor a root is connected to exactly three other processors in its tree: a parent and two children. Each root in a row or column tree is connected to its two children. Notice that in the mesh-of-trees the processors in each row and in each column can be looked upon as placed at levels $0, 1, \cdots, \log N^{1/2}$, where a level l, $0 \leq l \leq \log N^{1/2}$, has 2^l processors.

There is a natural $(3, 4(1 + \log N^{1/2}))$-embedding of the mesh-of-trees of base size N onto the reconfigurable mesh of size N. First, map the base mesh of the mesh-of-trees directly onto the reconfigurable mesh. Next, associate each tree node above a row or column with a base node of that row or column, using a mapping such that the ith processor on the lth level of the tree is mapped to the processor $i*2^{\log N^{1/2}-l}+2^{(\log N^{1/2})-1}$ of the row (or the column) [21], [22]. Notice that every processor in the reconfigurable mesh is responsible for at most three nodes of the mesh-of-trees, namely, a base node, a node from that base node's row tree, and a node from that base node's column tree. To simulate the communication links of the mesh-of-trees, four sets of subbuses are needed to simulate the base connections. Within each row and column there are $\log N^{1/2}$ levels of parent-child links. These are simulated level by level, using two sets of subbuses per level (for example, in rows, one set simulates connecting westernmost children to their parent and the other set simulates connecting easternmost children). The row and column switch settings are done separately, giving a total of $4(1 + \log N^{1/2})$ sets of settings.

Notice that this embedding is such that if a mesh-of-trees algorithm has the property that each communication step involves only communication within the base or only communication between processors at levels i and $i + 1$ for a given

i, then the algorithm can be converted into an normalized algorithm with a running time on the reconfigurable mesh that is at most a constant multiple slower than the original mesh-of-trees algorithm.

Corollary 4: Given a mesh-of-trees algorithm $\mathcal{A}$ taking $T(N)$ time steps on a mesh-of-trees of base size N on the reconfigurable mesh of size N:

1) $\mathcal{A}$ can be simulated in $O(T(N)\log N)$ time on the unit-time delay model and in $O(T(N)\log^2 N)$ time on the log-time delay model and
2) if $\mathcal{A}$ is normalized it can be simulated in $O(T(N))$ time on the unit-time delay model and in $O(T(N)\log N)$ time on the log-delay model.

□

A *pyramid computer (pyramid) of size* N is a machine that can be viewed as a full, rooted, 4-ary tree of height $\log_4 N$, with additional horizontal links so that each horizontal *level* is a mesh. It is often convenient to view the pyramid as a tapering array of meshes. A pyramid of size N has at its base a mesh of size N and a total of $\frac{4}{3}N - \frac{1}{3}$ processors. The levels are numbered so that the base is level 0 and the apex is level $\log_4 N$. A processor at level i is connected via bidirectional unit-time communication links to its nine neighbors (assuming they exist): four siblings at level i, four children at level $i-1$, and a parent at level $i+1$.

A (2,16)-embedding of the pyramid of base size N onto a reconfigurable mesh of size N can be generated by using the standard H-tree embedding of the layers above the pyramid's base onto its base [23] and using the natural map of the pyramid's base onto the reconfigurable mesh. Using this, one can partition the pyramid communication links into four sets, namely the mesh edges of the base, the mesh edges of all levels above the base, the parent-child edges connecting each even layer with the layer above, and the parent-child edges connecting each even layer with the layer below. For each of these sets there is a natural planar layout resulting in each node having degree 4 or less. Each set can in turn be naturally mapped into four collections of subbuses, giving a total of 16 collections of subbuses to simulate all the pyramid connections. Since the number of interprocessor communication operations is a constant no matter how large the pyramid, to within a multiplicative constant there is no advantage in simulating normalized pyramid algorithms as opposed to unnormalized ones.

Corollary 5: Given a pyramid algorithm $\mathcal{A}$ taking $T(N)$ time on a pyramid of base size N, on a reconfigurable mesh of size N $\mathcal{A}$ can be simulated in $O(T(N))$ time on the unit-time delay model and in $O(T(N)\log N)$ time on the log-time delay model. □

B. Graph Algorithms

The first problem considered in this section is that of computing the connected components of an undirected graph with $N^{1/2}$ vertices. The graph is given as an adjacency matrix, where the (i,j)th entry is initially stored in processor $P_{i,j}$ of the reconfigurable mesh. Upon termination of the algorithm, every processor $P_{i,j}$ will know the component label of vertex i and vertex j. The algorithm that we use is based on the $O(\log N)$ time CRCW PRAM algorithm presented in [24], which assumes a less restrictive unordered-edge input. We assume the reader is familiar with the algorithm in [24] and explain how to convert its PRAM steps into steps for the reconfigurable mesh. Components are labeled by vertex numbers, where initially each vertex is labeled with its own number. During each of the $O(\log N)$ iterations, the PRAM algorithm exploits two fundamental operations to update component labels. These operations, along with reconfigurable mesh implementations, are now briefly described.

1) The first operation is called *shortcutting*, which consists of every vertex "connecting" itself to its grandparent. This operation can be implemented on the reconfigurable mesh by the following steps.
   a) At the start of each iteration of the algorithm, all PE's $P_{i,j}$ will know the current parent (i.e., label) of vertex i, denoted by either $parent(i)$ or $label(i)$.
   b) Use column broadcasts from every diagonal processor $P_{i,i}$ so that every processor $P_{k,m}$ knows the current parent of vertex m.
   c) The value $parent(parent(i))$ is broadcast in row i from processor $P_{i,parent(i)}$ so that all processors $P_{i,j}$, simultaneously for all rows i, know the grandparent of vertex i.
2) The second operation is called *hooking*, which consists of every vertex i that points to a root j (i.e., $parent(i) = j$ is a root) trying to hook the root j to a non-leaf node p such that $label(p) < j$. This can be performed similar to step 1.

Notice that the implementation of both operations requires a fixed number of reconfigurable bus operations. Therefore, after a $\Theta(1)$ time initialization step to determine the initial parent (label) of every vertex, shortcutting, hooking, and other simple $\Theta(1)$ time operations can be used at each of the $O(\log N)$ iterations to obtain the following.

Theorem 1: Given the adjacency matrix of an undirected graph with $N^{1/2}$ vertices, distributed so that the (i,j)th element of the matrix is stored in processor $P_{i,j}$ of a reconfigurable mesh of size N, the connected components of the graph can be determined in $O(\log N)$ time under the unit-time delay model and in $O(\log^2 N)$ time under the log-time delay model. □

The reconfigurable mesh can also be used to provide efficient solutions to some graph problems that assume unordered edges as input. As an illustration, consider labeling the connected components of a V vertex graph on a reconfigurable mesh of size N. The general component-labeling algorithm that we follow has been used for a variety of parallel architectures (c.f., [25]). Initially, each processor $P_{p,q}$ stores an arbitrary edge (i,j). At the end of the algorithm, processor $P_{p,q}$ stores the label of the component to which vertices i and j belong. Component labels again correspond to vertex numbers. Initially, each vertex is its own component. During each of the $O(\log N)$ iterations, components are systematically merged so that the number of active components is reduced by at least a factor of 2. Therefore, the algorithm requires $O(\log N)$ iter-

ations to merge all vertices into their connected components, where each iteration consists of the following two operations.

1) In the first operation, every component chooses as its new label the minimum label of any component it is connected to.
   a) On the reconfigurable mesh, this is implemented by the key reduction operation as in Proposition 6, using the component labels as keys and minimum as the binary associative operator.
2) The second operation compresses the equivalence classes (supercomponents) that were just set up so that all vertices in the same class receive the same label. (e.g., if in the previous step component C chooses B as its new label, component B chooses A as its new label, and component A chooses itself as its new label, then this step (re)labels all three of these components to A.)
   a) On the reconfigurable mesh, collect the l active labels to the northwest block of size l. This can be done using the algorithm associated with Lemma 2. In this northwest block, a mesh computer algorithm finishing in $O(l^{1/2})$ time will assign every active label the minimum label in its class.
   b) The vertices are then relabeled according to the updated labels by creating N/l copies of the labels determined in the northwest block, as discussed previously (c.f., the algorithm of Lemma 2 and then performing mesh-computer RAR's within each local size $l^{1/2} \times l^{1/2}$ copy of the updated labels.

Initially each component consists of a single vertex. Since the steps of the algorithm are implemented by the operations in Proposition 6 and Lemma 2, the first iteration can be completed in $O(V^{1/2} + \log N)$ time under the unit-time delay model and $O(V^{1/2} \log N + \log^2 N)$ time under the log-time delay model. During each iteration the number of components is reduced by at least half. Therefore, the number of components during the ith iteration, $1 \leq i \leq \log V$, is no more than $V/2^i$. Hence, the algorithm can be completed in $O(V^{1/2} + \log N \log V)$ time under the unit-time delay model and in $O(V^{1/2} \log N + \log^2 N \log V)$ time under the log-time delay model.

The strong similarities between component-labeling algorithms and minimal spanning-forest algorithms for parallel models of computation are well known. In particular, others have noted that small changes to a component-labeling algorithm for a parallel computer can give a minimal spanning-forest algorithm for the same computer [26], [27]. The changes mainly consist of choosing edges of minimal weight at each stage of the algorithm, rather than edges incident on vertices of minimal label.

Several graph properties can be deduced once a spanning tree of the graph is determined [28]. Therefore, the following results hold.

Theorem 2: Given N or fewer edges of a graph G with V vertices distributed with no more than one edge per processor on a reconfigurable mesh of size N, in $\Theta(V^{1/2} + \log N \log V)$ time under the unit-time delay model and in $\Theta(V^{1/2} \log N + \log^2 N \log V)$ time under the log-time delay model, one can

1) label the components of G,
2) compute a minimal spanning forest of G,
3) check if G is bipartite,
4) compute the cyclic index of G, and
5) compute the articulation points of G.

□

C. Image Algorithms

Many problems involving digitized images have been solved on mesh-connected computers [47]. These problems can be solved efficiently on the reconfigurable mesh. The input for these problems is an $N^{1/2} \times N^{1/2}$ digitized image distributed one pixel per processor on a reconfigurable mesh of size N so that processor $P_{i,j}$ stores pixel (i, j). The pixels are either black or white where the interpretation is a black image on a white background.

The problems that we examine focus on labeling *figures (connected components)* and determining properties of the figures. The reconfigurable bus plays an important role in image algorithms, since it can be used to create a subbus within every figure so that information can be extracted about all figures concurrently. Indeed, a multilevel system for image understanding is being built, with a configurable bus system for low-level image processing [29], [30]. The first result of this section provides an algorithm to efficiently label the figures of an image.

Theorem 3: Given an $N^{1/2} \times N^{1/2}$ digitized image mapped one pixel per processor onto the processors of a reconfigurable mesh of size N, in $\Theta(\log N)$ time under the unit-time delay model and in $\Theta(\log^2 N)$ time under the log-time delay model, the figures (connected components) can be labeled.

Proof: In parallel, every processor examines the pixel in each of its four neighbors and sets its four switches so that a connection is maintained only between neighboring black pixels. This $\Theta(1)$ time operation creates a subbus over each figure. Given a linked list of processors overlayed by a reconfigurable subbus, the minimum (maximum) of the values stored in these processors can be computed in $O(\log N)$ iterations. Each iteration computes the local minima (maxima) and discards the other elements. Each iteration uses a constant number of broadcast steps and comparison operations, and therefore the times are as claimed. □

An alternative technique can be used if the concurrent write model for the bus can be used. Using subbuses in parallel a unique label can now be assigned to every figure by a standard bit-polling algorithm, as follows. Initially, all processors with a black pixel are *active*. At the ith iteration, all active processors with a 1 in the ith bit of their unique processor index, send a "1" to the bus.

If no processor sends a "1" to the bus, then the algorithm proceeds to the next iteration, while if at least one processor sends a "1" to the bus, then only those processors that sent a "1" remain active. During the ith iteration, every processor with a black pixel records the ith bit of its final component label, which is a 1 if a "1" is read from the bus, and is a 0 otherwise.

The next result is concerned with determining for each figure a nearest figure. The *distance between figures F and*

G is $\min\{d(f,g)|f \in F, g \in G\}$, where d is the Euclidean distance. For all results concerning distance or convexity, each pixel is treated as being an integer lattice point, rather than a small square.

Theorem 4: Given an $N^{1/2} \times N^{1/2}$ digitized image mapped one pixel per processor onto the processors of a reconfigurable mesh of size N, in $O(\log N)$ time under the unit-time delay model and in $O(\log^2 N)$ time under the log-time delay model, a nearest figure to each figure can be determined.

Proof: In the following algorithm we denote a processor *marked* if it has a "1" and at least one of its neighboring processors has a "0." The nearest neighbor algorithm operates in two phases. In the first phase, each marked processor of each figure locates a nearest processor to itself having a different label. In the second phase, a unique nearest figure to each figure is computed by a bottom-up merge.

In the first phase each marked processor can gather the information regarding its nearest "1" with a different label as follows: the row buses are configured into subbuses such that each subbus has processors with a "1" at its endpoints and processors storing "0"'s as intermediate nodes. Using broadcast, each processor with a "0" receives the indices of the endpoints of the row subbus to which it is connected. Repeating this along the columns, each processor with a "0" has four indices at this time. Each processor with a "0" computes the minimum of these values. All processors within a row (column) subbus can compute the minimum value stored in that subbus in $O(\log N)$ iterations by computing the local minima and discarding other elements. During each iteration, each processor of the subbus compares its value to that of its bus neighbor and refrains itself from broadcast in the succeding iterations if its value is greater than those stored in its bus neighbors.

In the second phase a bottom-up approach is used to compute the nearest figure to each label. The marked processors of a figure that belong to two adjacent blocks will compute the nearest figure to their label to yield a unique nearest figure to each label within the new block. This block merge can be performed by modifying the technique used in Lemma 4. On a $k \times k$ reconfigurable array, given k records in a column with each record having a label L_i, $1 \leq i \leq k$, and a value, the minimum (maximum) value within each label L_i can be computed in $O(1)$ time in the unit-time delay model for all labels in parallel.

Both phases can be performed in $O(\log N)$ steps, and therefore the times are as claimed. □

As we have shown, the reconfigurable mesh is particularly useful when the amount of essential data remaining can be rapidly reduced. For images, one common form of data reduction is to represent a figure by its *extreme points*, that is, by the corners of the smallest convex polygon containing the figure. It is particularly useful to *enumerate* the extreme points. A standard enumeration scheme is to start with the easternmost northernmost extreme point and number the extreme points in counterclockwise order. Enumerated extreme points store not only their number, but also the numbers and locations of the preceding and following extreme points.

Theorem 5: Given an $N^{1/2} \times N^{1/2}$ digitized image mapped one pixel per processor onto the processors of a reconfigurable mesh of size N, in $O(\log^2 N)$ time under the unit-time delay model and in $O(\log^3 N)$ time under the log-time delay model, the extreme points of the convex hull can be enumerated for every figure.

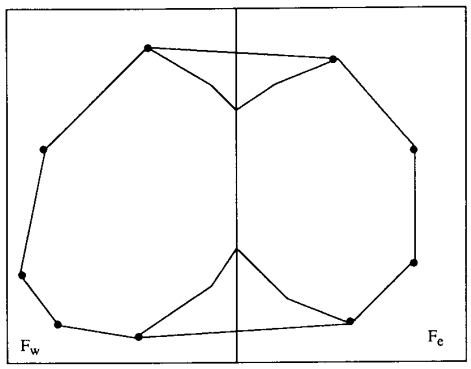


Fig. 6. Tangent lines for combining convex hulls.

Proof: We use a bottom-up divide-and-conquer approach. Assume that the image has been partitioned down the center and that in each half the extreme points of each figure restricted to that half have been enumerated. We show how to use this to find the extreme points of the entire figure. Note that this is only needed for figures crossing the center, for the others are finished.

For each figure F crossing the center, let F_w denote the portion of F on the western half, and let F_e denote the portion of F on the eastern half. We will locate the top and bottom tangent lines, shown in Fig. 6, and their four points of intersection with F. Extreme points of F_w and F_e are extreme points of F if and only if they do not lie within this quadrilateral. The process of locating these two tangent lines is similar, so we will only explain finding the northernmost one, denoted L. Let p denote the westernmost point of intersection of F and L, respectively. We will show how to locate p, with a similar procedure used to locate the easternmost point of intersection of F and L.

Let e be an edge of the convex hull of F_w and l_e a line collinear with e. If all points of F_e lie on the same side of l_e as F_w does, then p precedes the second endpoint of e in the counterclockwise ordering, while if some points of F_e lie on the opposite side of l_e, as F_w does, then p follows the first endpoint of e. This fact can be used to generate a binary-search procedure to find p. Each iteration of the binary search consists of broadcasting an oriented edge of the convex hull of F_w and then determining if any points of F_e lie on the opposite side. (An oriented edge gives the equation of the line and a normal vector toward the halfplane containing F_w) Notice that each iteration of the binary search reduces the candidate edges remaining by half. The broadcasting from F_w is done by the first extreme point of the edge. For the response from F_e, we have to insure that only one processor responds. Suppose a processor P_r contains the extreme point of F_e such that a line parallel to l_e passing through the extreme point in P_r will have all the rest of F_e on the side determined by the given

normal to e. Then P_r is responsible for responding, since all of F_e lies on the proper side of e if and only if the extreme point stored in P_r does. (It is possible that there is an edge of F_e parallel to l_e and having F_e on the proper side, in which case the processor storing the first endpoint of the edge in the enumeration is responsible for responding.)

The binary search to locate p, and each of the other three points of intersection of the tangent lines with F, takes $O(\log N)$ iterations, each of which takes $\Theta(1)$ time on the unit-delay model. Once the four extreme points are known, in $\Theta(1)$ time the extreme points in the quadrilateral formed by these points can be eliminated, and the extreme points can be renumbered. Therefore, in the unit-delay model the time obeys the recurrence $T(N) = T(N/2) + O(\log N)$, giving the times claimed. □

We now turn our attention to problems related to a single set of black pixels. We first show how two disjoint convex hulls can be merged on a reconfigurable mesh, and then recursively use this in our main result that gives efficient algorithms to enumerate the extreme points, determine the diameter, determine a smallest enclosing box, and determine the smallest enclosing circle of a set of pixels.

Lemma 3: Suppose S_1 and S_2 are sets of extreme points representing two disjoint convex hulls, $| S_1 | = | S_2 | = k$. Given S_1 and S_2 each stored one point per processor in a row of a $k \times k$ reconfigurable mesh, the extreme points of $S_1 \cup S_2$ can be determined in $O(1)$ time under the unit-time delay model and in $O(\log k)$ time under the log-time delay model.

Proof: Let the extreme points be stored in processors $P_1 \cdots P_k$ (using snake-like indexing of the processors) such that the ith processor stores the ith extreme point of S_1 and the ith extreme point of S_2. The algorithm relies on being able to find the two tangent lines from a point A outside of S_1 to the convex hull defined by S_1 in $O(1)$ time using the unit-time delay model and in $O(\log k)$ time using the log-time delay model. Suppose that each processor $P_1 \cdots P_k$ knows the point A. Then, each such processor P_i computes the angle formed by the x-axis and the line through the ith point of S_1 and A. Let w be the index of the processor having the westernmost southernmost extreme point. The angles in processors $P_1 \cdots P_w$ form a bitonic sequence. Also, a bitonic sequence is formed by the angles in processors $P_w \cdots P_k$. Thus, a processor P_m having the maximum (minimum) angle can easily identify itself by examining the contents of processors $P_{m\pm1}$. Notice that the minimum and the maximum angles correspond to the tangent lines from A to the convex hull defined by S_1.

Using the above idea, the two convex hulls defined by S_1 and S_2 can be merged as follows. Initially, the points in S_1 are stored one per processor in $P_{0,j}$, $0 \le j \le k-1$, as are the points of S_2.

1) Copy S_1 and S_2 to each row of the mesh. This is done for each set by a column broadcast.
2) Broadcast the ith point of S_2 to all processors in the ith row of the mesh.
3) Processor $P_{i,j}$ computes the angle formed between the ith point of S_2 and the jth point of S_1 with respect to the x-axis. The maximum and the minimum angles in each row are then detected by comparing angles in neighboring processors. Let $a_{\min_i}$ and $a_{\max_i}$ denote the the minimum and the maximum angles computed in the ith row. Using a broadcast in each row, these values can be stored in $P_{i,0}$, $0 \le i \le k-1$.
4) The common tangents to the two sets of extreme points are $\max(a_{\min_i})$ and $\min(a_{\max_i})$, which are computed using Proposition 3.

Since each step of the algorithm requires a fixed number of computations and bus operations, the running time is as claimed. □

The following theorem shows how to enumerate the extreme points of a single arbitrary set S of pixels and then uses the extreme points to determine additional properties of S. The *diameter* of S is $\max\{d(p,q) | p, q \in S\}$, where d is the Euclidean distance. A *smallest enclosing box* of S is a (not necessarily unique) rectangle of minimal area that contains S, and the *smallest enclosing circle* of S is the circle of minimal area that contains S.

Theorem 6: Given an $N^{1/2} \times N^{1/2}$ digitized image mapped one pixel per processor onto the processors of a reconfigurable mesh of size N, in $\Theta(1)$ time under the unit-time delay model and in $\Theta(\log N)$ time under the log-time delay model, several geometric properties of a (not necessarily connected) set S of pixels can be determined. These properties include marking and enumerating the extreme points of the convex hull of the points, determining the diameter of the points, determining the smallest enclosing box, and determining the smallest enclosing circle of the points.

Proof: These algorithms are based on being able to quickly reduce the N pieces of data to $O(N^{1/2})$ pertinent pieces of data from which the solution can be obtained. First we mark and enumerate the extreme points, and then use these points to solve the remaining problems.

The algorithm for finding and enumerating the extreme points of a given set S of pixels (distributed over a $N^{1/2} \times N^{1/2}$ mesh) is as follows.

1) Let w_i and e_i denote the westernmost and easternmost points in S in the ith row, $0 \le i \le N^{1/2} - 1$. Each of these can be identified by a bus-splitting operation.
2) Divide the mesh into disjoint *row blocks*, each of size $N^{1/4} \times N^{1/2}$. Each such row block has a subset of points identified in step 1, which consists of no more than $2N^{1/4}$ points. Let L_i and R_i denote the sets of westernmost and easternmost points, respectively, located in the ith row block. Compute the convex hull of L_i and R_i using the $N^{3/4}$ processors of the ith row block as follows.
   a) Divide each row block into disjoint subblocks of size $N^{1/4} \times N^{1/4}$. Using a row broadcast, each subblock of the ith row block can store L_i and R_i.
   b) Use the jth subblock to decide whether or not w_j is an extreme point of L_i. Consider w_j as the origin and determine for both the upper and lower halfplanes the points of L_i located at the minimum and maximum angles relative to w_j. Then w_j is an extreme point of L_i if and only if it is not contained in the convex hull of these four points.

These minimums and maximums can be determined by using Proposition 3. The extreme points of L_i can be enumerated using Corollary 1. The extreme points of R_i are computed similarly.

3) Using row and column broadcasts, the resulting sets L_i^* and R_i^* of extreme points in the ith row block are moved to each subblock of the ith row block and to the ith subblock of each row block. Using Lemma 3, in block (i, j) the common tangent lines between L_i, L_j, R_i, and R_j can be computed.
4) The previous step results in $O(N^{1/4})$ pairs of tangent lines in each row block. Compute the slope of these lines with the x-axis and compute the minimum and the maximum such angle using Proposition 3. From this, the extreme points of L_i^* and R_i^*, which are the extreme points of S, can be determined. These points can be enumerated using Corollary 1.

The algorithm uses a constant number of broadcast and bus-splitting operations and thus the time is as claimed.

The diameter of the set of black pixels is easily determined once the extreme points have been marked. To do this, for each extreme point the maximum distance to any other extreme point is computed, and the maximum of these distances is the diameter. Note that the number of extreme points of a set of pixels in a grid of size $N^{1/2} \times N^{1/2}$ is $O(N^{1/3})$. To compute the farthest pixel to each extreme point, the mesh is partitioned into $N^{1/3}$ disjoint blocks of size $N^{1/6} \times N^{1/2}$ each. Using broadcast operations, the ith extreme point can be moved to the ith block and distributed to all processors in the block. Using broadcast operations, the extreme points of S are moved to each row of the mesh, residing in the first $N^{1/3}$ columns. Within the ith block, divide the portion in the first $N^{1/3}$ columns into $N^{1/6}$ subsquares of size $N^{1/6} \times N^{1/6}$. Within each subsquare there are $N^{1/6}$ extreme points. Use Proposition 3 to find the maximum distance from any of these to the ith extreme point. Now within the ith block there are $N^{1/6}$ distances, one per square. Move these to the leftmost square, and again determine the maximum. This is the maximum distance from the ith extreme point to any other extreme point. The maximum of the resulting $O(N^{1/3})$ distances is the diameter and it can be computed using Proposition 3. Each stage takes constant time in the unit-delay model, or $O(\log N)$ time in the log-delay model, giving the time as claimed.

To compute a smallest enclosing box, use the fact that a smallest enclosing box has one of the edges of the convex hull collinear with one of its sides [31]. The other sides of the box can be determined by computing the extreme point tangent to each side of the box. Use row and column broadcasts so that every processor $P_{i,j}$ contains hull edge $e_{i,j}$. Then, each column i will be used to compute a smallest box that contains an edge collinear with e_i by determining points tangent to the other three sides of the box. This can be done by bus-splitting operations since the distances of the extreme points from the line collinear with the hull edge form a bitonic sequence along the column responsible for the edge. Finally, the minimum of these $O(N^{1/3})$ values is computed.

A smallest enclosing circle, its center, and its radius, can be efficiently computed by the algorithm described below.

1) Divide the mesh into blocks of size $N^{1/2} \times N^{1/6}$.
2) The image is divided into $N^{1/3}$ disjoint squares of size $N^{1/3} \times N^{1/3}$. Each square is represented by the coordinates of its center. For each square the maximum of the distance between center of the square to an extreme point is computed. Denote this as d_m. The square in which the center of the smallest enclosing circle lies is either the one having minimum d_m or one of its eight adjacent squares. The computations regarding the distances of the extreme points to the center of a square are done in the same manner as was done above for determining the diameter.
3) The nine squares determined in the above step are divided into $9N^{1/3}$ disjoint subsquares of size $N^{1/6} \times N^{1/6}$, and the process in step 1 is repeated for these subsquares, finding the one with the minimal maximal distance to any extreme point, and using it and its eight neighbors for the next state.
4) Following the same reasoning, among the $9N^{1/3}$ pixels of the squares chosen in step 2, the center of the smallest enclosing circle is the pixel having minimum d_m. The radius of the circle is the d_m of the center.

The computation of each step takes constant time in the unit-delay model and $O(\log N)$ time in the log-delay model, so the total time is as claimed. □

V. Conclusion

This paper has introduced the reconfigurable mesh as a model of computation that captures salient features common to a number of reconfigurable architectures. We have presented efficient fundamental data-movement operations, as well as efficient solutions to fundamental problems. In fact, we have shown that the reconfigurable mesh is more powerful than the PRAM for certain problems, including EXOR computation. We have shown how to embed other parallel architectures onto the reconfigurable mesh and have presented efficient solutions to graph and image problems that rely on the fundamental data-movement operations.

Several of the algorithms presented here can be shown to be optimal. For example, any problem solved in $\Theta(1)$ time on the unit-time delay model or in $\Theta(\log N)$ time on the log-time delay model is optimal. For the unit-time delay model this is obvious, and for the log-time delay model it comes from noting that each problem may require combining data originating arbitrarily far apart and that this takes $\Omega(\log N)$ time on the log-time delay model. Further, the unit-time delay algorithm in Proposition 4 is optimal by the results in [18], under the assumption that the only allowable operation on the data items is to compare them. However, for many of the problems considered in this paper, the optimality of our results remains an open problem. For some of the problems we are aware of slightly faster, though more complicated, algorithms. This is particularly true of results for the log-time delay model.

If one considers simulating the reconfigurable mesh on other architectures, then it can be shown that for an algorithm that

runs in $T(N)$ time on a reconfigurable mesh of size N under the unit-delay model,

- the mesh of size N can simulate the algorithm in $O(T(N)N^{1/2})$ time,
- the pyramid of base size N can simulate the algorithm in $O(T(N)N^{1/4})$ time,
- the mesh-of-trees of base size N can simulate the algorithm in

$$O(\frac{T(N)\log^3 N}{\log\log N})$$

time,
- the hypercube of N processors can simulate the algorithm in $O(T(N)\log^2 N)$ time, and
- the CRCW PRAM of N processors can simulate the algorithm in $O(T(N)\log N)$ time.

These simulations are based on reducing the simulation problem to that of labeling the connected components of connected sets of base processors at every step of the step-by-step simulation. (Component labeling algorithms found in [3], [32], [33] provide the solution.) In fact, if the reconfigurable mesh algorithm only sets the switches once at the beginning of the algorithm, then after an initial component labeling phase, the stepwise simulation time of the mesh-of-trees, hypercube, and PRAM simulations can be reduced by a logarithmic factor since the message passing is now reduced to broadcasting a value throughout a labeled component, instead of labeling the components at every step of the simulation. Some of the simulations can be shown to be optimal using information transfer arguments (for definition of information transfer and techniques to estimate it, see [34], [35]).

We gave a rather general simulation result, showing that, given an architecture with an embedding that is somewhat similar to a multilevel layout, the reconfigurable mesh can do an efficient stepwise simulation. As examples, we used this to show that the reconfigurable mesh could efficiently simulate the pyramid and mesh-of-trees. It is easily seen that any architecture that can be laid out in an $N^{1/2} \times N^{1/2}$ grid (assuming wires have unit width) using only a single wire layer, can be simulated on the unit-time delay reconfigurable mesh in constant time per unit time of the target architecture.

A number of researchers have been interested in the reconfigurable-mesh model after we presented the ideas in this paper in conferences. Additional results can be found in [36]–[44], [49]. Finally, this paper was motivated by a number of parallel systems being built that provide reconfigurable connections between processors. New advances in holography and other optical techniques offer certain advantages compared to traditional electronic systems that can be exploited to provide reconfigurable interconnections [45].

REFERENCES

[1] C. R. Dyer, "A VLSI pyramid machine for hierarchical parallel image processing," in *Proc. IEEE Conf. Pattern Recognition Image Processing*, 1981.

[2] S. L. Tanimoto, "A pyramidal approach to parallel processing," in *Proc. Int. Symp. Comput. Architecture*, June 1983, pp .

[3] R. Miller and Q. F. Stout, "Data movement techniques for the pyramid computer," *SIAM J. Comput.*, vol. 16, no. 1, Feb. 1987.

[4] F. T. Leighton, "Parallel computations using mesh of trees," MIT, Cambridge, MA, Tech. Rep., 1982.

[5] D. Nath, F. N. Maheshwari, and P. C. P. Bhatt, "Efficient VLSI networks for parallel processing based on orthogonal trees," *IEEE Trans. Comput.*, 1983.

[6] Q. F. Stout, "Mesh connected computers with broadcasting," *IEEE Trans. Comput.*, vol. C-32, pp. 826–830, 1983.

[7] S. H. Bokhari, Finding maximum on an array processor with a global bus, *IEEE Trans. Comput.*, vol. C-33, no. 2, pp. 133–139, Feb. 1984.

[8] V. K. Prasanna-Kumar and C. S. Raghavendra, "Array processor with multiple broadcasting," in *Proc. Annu. Symp. Computer Architecture*, June 1985.

[9] Q. F. Stout, "Meshes with multiple buses," in *Proc. 27th IEEE Symp. Foundations Comput. Sci.*, 1986, pp. 264–273.

[10] Hunt, "The ICL DAP and its application to image processing," in *Languages and Architectures for Image Processing*, M. J. B. Duff and S. Levialdi, Eds. New York/London: Academic Press, 1981.

[11] L. Snyder, "Introduction to the configurable highly parallel computer," *Comput.*, vol. 15, no. 1, pp. 47–56, Jan. 1982.

[12] A. Aggarwal, "Optimal bounds for finding maximum on array of processors with k global buses," *IEEE Trans. Comput.*, vol. C-35, no. 1, pp. 62–64, Jan. 1986.

[13] D. M. Champion and J. Rothstein, "Immediate parallel solution of the longest common subsequence problem," in *Proc. Int. Conf. Parallel Processing*, Aug. 1987, pp. 70–77.

[14] H. Li and M. Maresca, "Polymorphic-torus network," *IEEE Trans. Comput.*, vol. 38, no. 9, pp. 1345–1351, Sept. 1989.

[15] J. D. Ullman, *Computational Aspects of VLSI*. New York: Computer Science Press, 1984.

[16] V. K. Prasanna-Kumar and D. Reisis, "Parallel image processing on enhanced arrays," in *Proc. Int. Conf. Parallel Processing*, Aug. 1987, pp. 909–912.

[17] M. Furst, J. Saxe, and M. Sipser, "Parity, circuits and polynomial time hierarchy," in *Proc. IEEE Symp. Found. Comput. Sci.*, Oct. 1981, pp. 260–270.

[18] L. G. Valiant, "Parallelism in comparison problems," *SIAM J. Comput.*, vol. 3, 1975.

[19] R. E. Ladner and M. J. Fischer, "Parallel prefix computation," *J. Assoc. Comput. Mach.*, vol. 27, pp. 831–838, 1980.

[20] D. Nassimi and S. Sahni, "Data broadcasting in SIMD computers," *IEEE Trans. Comput.*, vol. C-30, no. 2, pp. 101–107, Feb. 1981.

[21] V. K. Prasanna-Kumar and D. Reisis, "VLSI arrays with reconfigurable buses," Comput. Res. Inst., Univ. Southern CA, Los Angeles, Tech. Report CRI-87-48, Sept. 1987.

[22] R. Miller and Q. F. Stout, "Some graph and image processing algorithms for the hypercube," in *Proc. SIAM Conf. Hypercube Multiprocessor*, 1987.

[23] Q. F. Stout, "Pyramid computer algorithms optimal for the worst-case," in *Parallel Computer Vision*, L. Uhr, Ed. New York: Academic Press, 1987, pp. 147–168.

[24] Y. Shiloach and U. Vishkin, "A $O(\log N)$ parallel connectivity algorithm," *J. Algorithms*, vol. 3, 1982.

[25] D. S. Hirschberg, A. K. Chandra, and D. V. Sarwate, "Computing connected components on parallel computers," *Commun. Assoc. Comput. Mach.*, pp. 461–464, 1979.

[26] S. E. Hambrusch and J. Simon, "Solving undirected graph problems on VLSI," Dept. Comput. Sci., PA State Univ., Univ. Park, PA, Tech. Rep. CS-81-23, 1981.

[27] C. Savage and J. Ja'Ja', "Fast, efficient parallel algorithms for some graph problems," *SIAM J. Comput.*, vol. 10, pp. 682–691, 1981.

[28] M. Atallah and R. Kosaraju, "Graph problems on a mesh connected processor array," *J. Assoc. Comp. Mach.*, vol. 31, pp. 649–667, 1983.

[29] J. G. Nash and D. B. Shu, "The image understanding architecture," in *Proc. 21st Annu. Asilomar Conf. Signals, Syst., Comput.* (Monterey, CA), Nov. 1987.

[30] C. C. Weems *et al.*, "The image understanding architecture," *Int. J. Comput. Vision*, vol. 2, pp. 251–282, 1989.

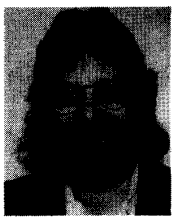
[31] H. Freeman and R. Shapira, "Determining the minimal-area enclosing rectangle for an arbitrary closed curve," *Commun. Assoc. Comput. Mach.*, vol. 18, pp. 409–413, 1975.

[32] V. K. Prasanna-Kumar and M. Eshaghian, "Parallel geometric algorithms for digitized pictures on mesh of trees organization," in *Proc. Int. Conf. Parallel Processing*, Aug. 1986, pp. 270–273.

[33] H. M. Alnuweiri and V. K. Prasanna-Kumar, "Parallel architectures and algorithms for image component labeling," Inst. Robotics Intell. Syst., Tech. Rep. IRIS #253, May 1989.

[34] J. Ja'Ja' and V. K. Prasanna-Kumar, "Information transfer in distributed computing with applications to VLSI," *J. Assoc. Comput. Mach.*, vol. 31, no. 1, pp. 150–162, Jan. 1984.

[35] J. Ja'Ja', V. K. Prasanna-Kumar and J. Simon, "Information transfer under different sets of protocols," *SIAM J. Comput.*, vol. 13, no. 4, pp. 840–849, Nov. 1984.
[36] J.-W. Jang and V. K. Prasanna, "An optimal sorting algorithm on reconfigurable mesh," Inst. Robotics Intell. Syst., Tech. Rep. IRIS#277, Aug. 1991.
[37] J-W. Jang, H. Park and V. K. Prasanna, "A fast algorithm for computing histogram on reconfigurable mesh," Inst. Robotics Intell. Syst., Tech. Rep. IRIS#290, Feb. 1992.
[38] J.-W. Jang, H. Park, and V. K. Prasanna, "An optimal multiplication algorithm on reconfigurable mesh," Inst. Robotics Intell. Syst., Tech. Rep. IRIS#294, Mar. 1992.
[39] D. Reisis, "An efficient convex hull computation on the reconfigurable mesh," in *Proc. Int. Parallel Processing Symp.*, 1992, pp. 142–145.
[40] Y. Ben-Asher, D. Peleg, R. Ramaswami, and A. Schuster, "The power of reconfiguration," *J. Parallel Distributed Computing*, pp. 139–153, 1991.
[41] J. Jenq and S. Sahni, "Reconfigurable mesh algorithms for the area and perimeter of image components and histogramming," in *Proc. Int. Conf. Parallel Processing*, 1991, pp. 280–281.
[42] J. Jenq and S. Sahni, "Reconfigurable mesh algorithms for image shrinking, expanding, clustering, and template matching," in *Proc. Int. Parallel Processing Symp.*, 1991, pp. 208–215.
[43] K. Nakano, T. Masuzawa, and N. Tokura, "A sub-logarithmic time sorting algorithm on a reconfigurable mesh," *IEICE Trans.*, vol. E74, no. 11, pp. 3894–3901, Nov. 1991.
[44] B. F. Wang, G. H. Chen, and F. C. Lin, "Constant time sorting on a processor array with a reconfigurable bus systems," *Info. Processing Letts.*, pp. 187–192, 1990.
[45] M. M. Eshaghian and V. K. Prasanna-Kumar, "VLSI electro-optical computers for signal and image processing," in *Proc. 3rd Int. Conf. Supercomputing*, 1988.
[46] R. Miller, V. K. Prasanna-Kumar, D. Reisis, and Q. F. Stout, "Meshes with reconfigurable buses," in *Proc. MIT Conf. Advanced Res. VLSI*, Jan. 1988, pp. 163–178.
[47] R. Miller and Q. F. Stout, "Geometric algorithms for digitized pictures on a mesh-connected computer," *IEEE Trans. Pattern Analysis Mach. Intell.*, vol. PAMI-7, pp. 216–228, 1985.
[48] D. Reisis and V. K. Prasanna-Kumar, "VLSI arrays with reconfigurable buses," in *Proc. Int. Conf. Supercomputing* (Athens, Greece), June 1987.
[49] S. Olariu, J. Schwoing, and J. Zhang, "Fast Computer Vision Algorithms for Reconfigurable Meshes," in Proc. Int. Parallel Processing Symp., Mar. 1992, pp. 258–261.



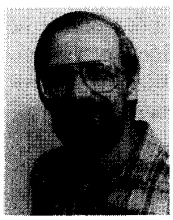
Russ Miller (S'84–M'85) was born in Flushing, New York, on January 8, 1958. He received the B.S., M.A., and Ph.D. degrees in computer science/mathematics from the Department of Mathematical Sciences, State University of New York, Binghamton, in 1980, 1982, and 1985, respectively.

In 1985, he became an Assistant Professor in the Department of Computer Science at the State University of New York, Buffalo, and has been an Associate Professor since 1990. He currently serves as a consultant for Thinking Machines Corporation and for three years served as Associate Director for the Graduate Group in Advanced Scientific Computing at the State University of New York, Buffalo. His primary research interests are parallel algorithms, parallel computing, parallel architectures, and computational crystallography. He has coauthored over 50 technical papers in these areas.

Dr. Miller is a member of the IEEE Computer Society, the Association for Computing Machinery, SIAM, and Phi Beta Kappa. He is also on the editorial board of *Parallel Processing Letters*.

V. K. Prasanna (V. K. Prasanna-Kumar) (M'84–SM'91), for a photograph and biography please see page 642 of this issue.

Dionisios I. Reisis, photograph and biography not available at time of publication.



Quentin F. Stout (M'82–SM'92) was born on September 23, 1949. He received the B.A. degree from Centre College, Danville, Kentucky, in 1970, and the Ph.D. degree from Indiana University in 1977.

Since 1984, he has been an Associate Professor in the Electrical Engineering and Computer Science Department at the University of Michigan, Ann Arbor, where he is a member of the Advanced Computer Architecture Laboratory and the Laboratory for Scientific Computing. From 1976 to 1984 he was on the faculty of the State University of New York, Binghamton. His primary research interests are in parallel computing, especially parallel algorithms. He recently coedited (with Hungwen Li) *Reconfigurable Massively Parallel Computers*, published by Prentice-Hall.

Dr. Stout is a member of the IEEE Computer Society, the American Mathematical Society, the Association for Computing Machinery, and the Society for Industrial and Applied Mathematics. He is on the editorial board of the IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS.