JOURNAL OF PARALLEL AND DISTRIBUTED COMPUTING **44,** 133–140 (1997)
ARTICLE NO. PC971353

Time-Efficient Maze Routing Algorithms on Reconfigurable Mesh Architectures

F. Ercal[1] and H. C. Lee[2]

Computer Science Department and Intelligent Systems Center, University of Missouri-Rolla, Rolla, Missouri 65401

The routing problem is one of the most widely studied problems in VLSI design. Maze-routing algorithms are used in VLSI routing and robot path planning. Efficiency of the parallel maze routing algorithms which were mostly based on C. Y. Lee's algorithm (1961, *IRE Trans. Electron. Comput.* (Sept.), 346–365) is poor. In this paper, we propose time-efficient algorithms to solve the maze-routing problem on a reconfigurable mesh architecture. The constant-time algorithms presented include: (i) testing the existence of specific types of paths between two terminals, and (ii) finding an absolute shortest path (ASP) and a shortest duplex-path (SDP). In addition, a fast algorithm to find the single shortest path (SSP) is presented. The simulation results indicate that a large percentage of the shortest paths that exist between two randomly selected terminals fall into one of the categories studied in this paper.

1. INTRODUCTION

Cost of the interprocessor communications crucially affects the efficiency of parallel algorithms. Reconfigurable mesh (RMESH) architectures offer the needed efficiency and flexibility in interprocessor communications by allowing the network topology to change dynamically as required by the algorithm. Since the advances in VLSI technology have started offering hope for manufacturing such chips [1, 2] in recent years, researchers have shown an increased interest in designing efficient algorithms for such architectures [3–7].

The maze-routing problem can be simply defined as finding a shortest path between two terminals following the nonblocked cells in a mesh (see Fig. 1). VLSI routing and robot path planning are two important applications where maze-routing algorithms are used. Lee's algorithm [8] is the most popular sequential algorithm for maze routing and has a worst-case running time of $O(N^2)$ on a maze of dimension $N \times N$. A number of parallel versions of the Lee's algorithm have been proposed [9–11]. Since the wavefront expansion in Lee's algorithm is sequential in nature, the efficiency of the proposed parallel algorithms is limited. Because of the topology and reconfigurability, RMESH architectures are excellent candidates to solve certain restricted classes of the maze-routing problem efficiently.

This paper is organized as follows. Section 2 gives some of the fundamental terms and definitions used in the paper and also introduces the RMESH model. Section 3 presents several constant-time algorithms for a 2D maze-routing problem. Another time-efficient algorithm to find a uniquely available path between two terminals is given in Section 4. Simulation results to show the distribution of different types of paths are given in Section 5 which is followed by the conclusions.

2. PRELIMINARIES

2.1. Basic Definitions and Propositions

DEFINITION 2.1. *Manhattan distance* between two cells, a and b, whose coordinates are (x_a, y_a) and (x_b, y_b), respectively, is defined as $\text{Manhattan}(a, b) = |x_a - x_b| + |y_a - y_b|$.

DEFINITION 2.2. Given a maze with $N \times N$ cells, a source cell $S = (x_s, y_s)$ and a target cell $T = (x_t, y_t)$, the rectangle which has ST as its diagonal is called the *S–T rectangle*.

DEFINITION 2.3. A path between a pair of cells S and T is called an absolute shortest path between S and T (and is denoted by $ASP(S, T)$) iff the length of this path equals $Manhattan(S, T)$.

Figures 1a and 1b illustrate an absolute shortest path ($ASP(S, T)$) and the S–T rectangle for terminals S and T, respectively. Based on these definitions, we can prove the following:

PROPOSITION 2.1. *If there are no blocked cells inside the rectangular area defined by a source cell $S = (x_s, y_s)$ and a target cell $T = (x_t, y_t)$ as shown in Fig. 1b, then there are $\binom{m+n}{n}$ different ASPs between S and T where $m = |x_t - x_s|$ and $n = |y_t - y_s|$.*

Proof. For the sake of simplicity, let's assume that T is located in the southeast of S as shown in Fig. 1b. The length of any ASP from S to T is $(m + n)$ and can be defined in terms of "→" and "↓" arrows each representing one-hop in the path (refer to the representation of P_1 in Fig. 1b for an

[1]E-mail: ercal@cs.umr.edu, URL: http://www.cs.umr.edu/~ercal/
[2]E-mail: hlee@cs.umr.edu, URL: http://wwww.cs.umr.edu/~hlee/

0743-7315/97 $25.00

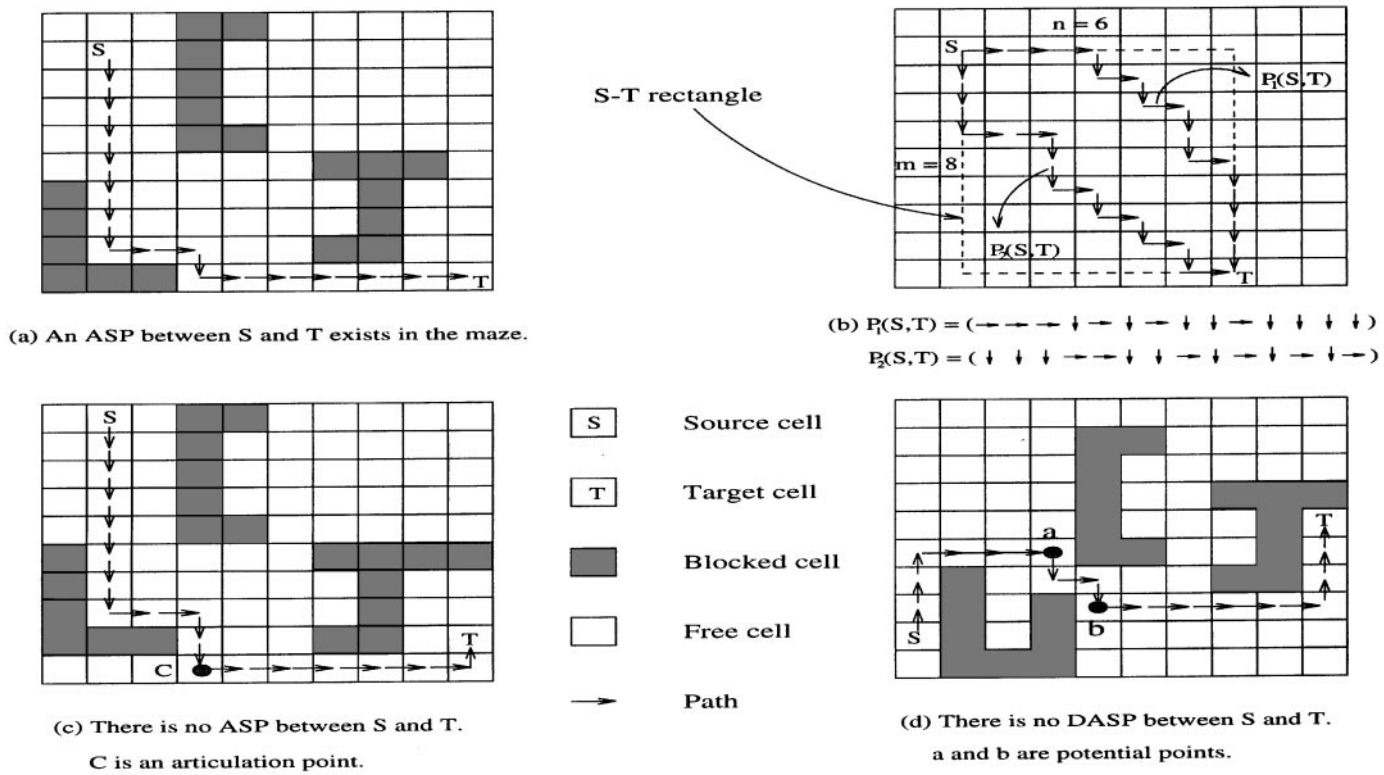

FIG. 1. Mazes with different blockages.

illustration). In any such path definition, there will be n "→"s and m "↓"s. Each arrangement of n "→"s (similarly, m "↓"s) in different locations defines a new ASP. Therefore, there are $\binom{m+n}{n}$ different potential ASPs between S and T. ■

PROPOSITION 2.2. *No cell in the path of an ASP*(S, T) *can be outside the S–T rectangle.*

Proof. Without loss of generality, let's assume that T is located in the southeast of S as shown in Fig. 1b. The length of any *ASP*(S, T) is $(m + n)$ and can be defined in terms of exactly n "→" and m "↓" arrows. In whatever order these n "→" and m "↓" arrows are placed, they cannot exit the S–T rectangle; i.e., all of the ASP cells will be inside the S–T rectangle or at the boundary. ■

2.2. RMESH Model

There are various but similar reconfigurable mesh (RMESH) architectures proposed in the literature. The algorithms presented in this paper use a 2D directional RMESH model as shown in Fig. 2. In this model, each PE has separate input and output ports corresponding to each neighbor which can be set (unset) individually. Some important features of this model are as follows:

1. A 2D RMESH is an $N \times N$ array of PEs connected in a standard mesh topology. The index of a PE is a 2-tuple (i, j), where $0 \le i \le N - 1$ is the row index and $0 \le j \le N - 1$ is the column index.

2. Each of the PEs has four sets of ports that are connected to its four nearest neighbors in the plane as shown in Fig. 2. $\{E_i, S_i, W_i, N_i\}$ stands for the set of input ports from the east, south, west, and north neighbors, respectively. $\{E_o, S_o, W_o, N_o\}$ is the list of output ports to the east, south, west, and north neighbors, respectively. Within a PE, any combination of the input ports can be connected to any combination of the output ports. This feature allows multiple busses to pass through a particular PE.

3. Output ports cannot read the bus, and input ports cannot write onto the bus. Data between PEs can move only from output ports into input ports.

4. A link is established if and only if the input (output) port and the corresponding output (input) port are both closed at the same time.

5. Only one processor in a subnet can write data on the bus at any given time. In unit time, data put onto a directional bus can be read by every PE who is reachable from the source PE by following the directional links that define the subnet.

The list of ports connected to the same bus within a PE will be indicated inside { } parentheses. For example, when we say, "a processor sets its $\{X_i, Y_i, Z_o\}$ switches," it means that the input ports, X_i and Y_i, and the output port, Z_o, are enabled and connected to the same bus within a PE. Figure 2b illustrates a case where processors set their $\{N_i, W_i, S_o,$

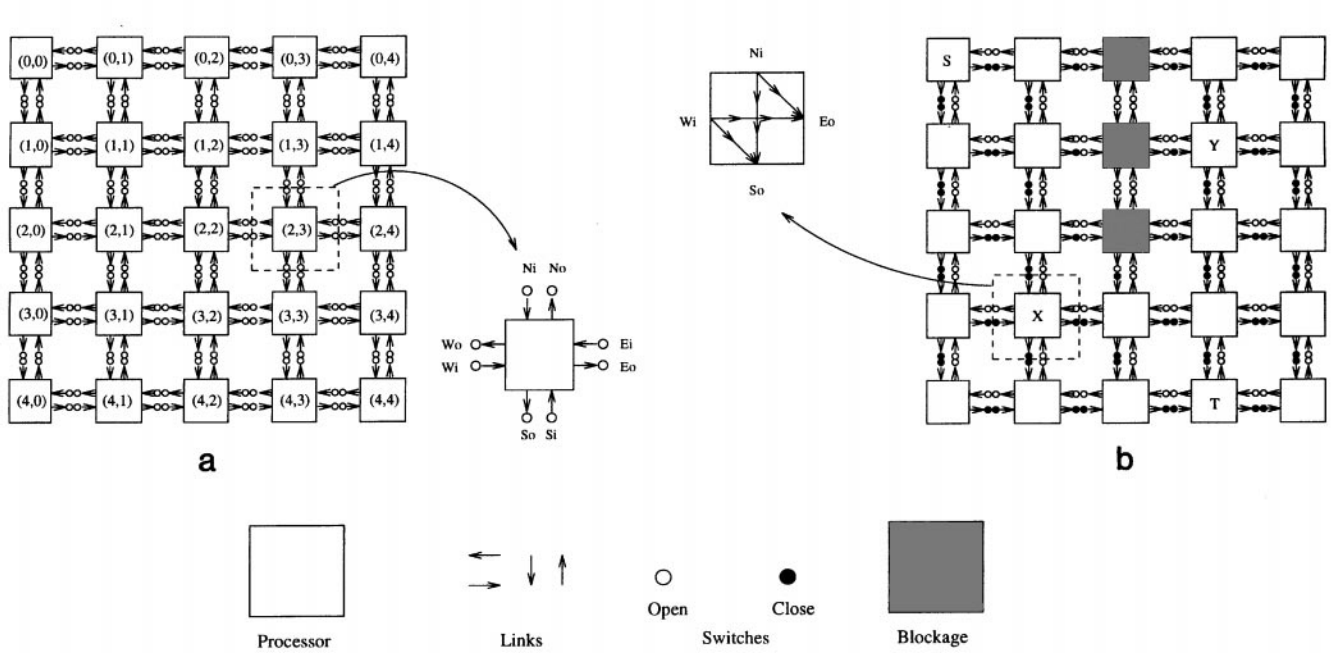

FIG. 2. A 2D RMESH with 4 × 4 processors.

E_o} switches. With this configuration, data written on the bus by processor *S* can be read by processors *X* and *T*, but not processor *Y*. Similarly, if *Y* writes on the bus, it can only be received by *T* but not *S* or *X*.

3. CONSTANT-TIME ALGORITHMS

Two major algorithms along with their relevant definitions are presented in this section: (i) finding an absolute shortest path (ASP) between two terminals, *S* and *T*, and (ii) finding a shortest duplex path (DSP) between *S* and *T*. Both algorithms run in constant-time on an $N \times N$ RMESH.

3.1. Finding an Absolute Shortest Path (ASP)

First, we will give an algorithm to check the existence of a path between *S* and *T*.

Algorithm 3.1 (Determine the existence of a path from *S* to *T*).

• Step 1: [Form a subnet which includes only the nonblocked cells] All the blocked PEs (those PEs representing a blocked maze cell) disable all of their input and output ports. The remaining PEs (nonblocked ones) enable all of their input and output ports.

• Step 2: The source PE (representing *S*) broadcasts a special signal (*s*) on the bus and all the other PEs including the target PE (*T*) read the bus. If *T* receives the signal *s*, then there exists a path between *S* and *T*.

Since all of the steps in Algorithm 3.1 can be performed in constant time, checking the existence of a path between two terminals on a maze has a running time of O(1). Once we verify the existence of a path between the source and target cells, we may proceed to find the shortest among all available paths. To the best of our knowledge, there is no known constant-time algorithm to find the shortest path on an RMESH. In this study, we attempt to find certain classes of shortest paths in constant-time. In other words, if the shortest path fits a certain definition, our algorithms are able to find it in constant-time on an RMESH architecture. Otherwise, we do not guarantee a constant-time algorithm for determining the shortest path between two terminals. First, we will describe an algorithm to find the ASP-Reachable Points and then proceed with the algorithm to find an ASP.

DEFINITION 3.1. Given a maze with $N \times N$ cells and a cell *Y*, if there exists an *ASP*(*Y*, *X*) then *X* is called an *ASP-Reachable Point* (*ARP*) of *Y*.

ALGORITHM 3.2 (FIND_ARP: Find all the ASP-Reachable Points (ARPs) of a cell Z).

• Step 1: Z broadcasts its coordinates (x_z, y_z) to all the PEs and they record it.

• Step 2: [Form special directional subnets] Each nonblocked PE(x_i, y_i) executes:

—Case 1: If $x_z \le x_i$ and $y_z \le y_i$, then set {N_i, W_i, S_o, E_o} switches.

—Case 2: If $x_z \le x_i$ and $y_z > y_i$, then set {N_i, E_i, S_o, W_o} switches.

—Case 3: If $x_z > x_i$ and $y_z \le y_i$, then set {S_i, W_i, N_o, E_o} switches.

—Case 4: If $x_z > x_i$ and $y_z > y_i$, then set {S_i, E_i, N_o, W_o} switches.

All blocked PEs open all their switches.

• Step 3: Z broadcasts a special signal z on the bus and all the other PEs read the bus.

• Step 4: Those PEs which receive z mark themselves as ARP of Z.

The lemma below immediately follows Algorithm 3.2.

LEMMA 3.1. *The existence of an absolute shortest path between a source cell $S(x_s, y_s)$ and a target cell $T(x_t\, y_t)$ can be determined in $O(1)$ time on an RMESH architecture.*

Proof. The existence of an ASP between a source (S) and a target cell (T) can be easily determined by using Algorithm 3.2 and checking if T is an ARP of S. It is also clear that running time of Algorithm 3.2 on an RMESH architecture is O(1). Hence, we only need to prove that Algorithm 3.2 is correct. If we represent an ASP using only arrows as shown in Fig. 1b, no two opposite directed arrows (such as "↑" and "↓," for example) can coexist in its path definition. Otherwise, it cannot be an ASP. This result works in both directions. In other words, if no two opposite arrows are present in a path definition, it represents an ASP. Now, we prove the lemma by showing two results: (a) if T receives the signal sent by S, then there exists at least one ASP between S and T, and (b) if there exists an ASP between S and T, T is guaranteed to receive the signal sent by S.

Proof for (*a*). In Algorithm 3.2, the input/output switches are set in such a way that there will be only two types of arrows (which are not opposite) present in the entire RMESH network. If T receives the signal sent by S, then there must be at least one directed path from S to T. Since the definition of any such path cannot contain two opposite arrows, it should be an ASP.

Proof for (*b*). If there exists an ASP between S and T, it can be represented by one or two types of arrows which are in the T direction. In Algorithm 3.2, switch settings are done in such a way that output arrows (at most two) originating from S always point in the direction of T and consistently move in the same direction. In other words, only these types of arrows are used in the entire subnet. Since each path which can be formed by using these types of arrows in the original maze will be a subset of the entire subnet formed by the RMESH PEs, T will be reachable from S. If no such path exists (see Fig. 1c), the signal sent by S will not be received by T. Therefore, a signal originating from S can only reach T iff there exists at least one ASP from S to T. ■

If T receives the signal sent by S, it implies that there exists at least one ASP. In general, there may be more than one ASP originating from S and leading to T. In the following, we describe an algorithm to mark the cells belonging to one particular ASP out of many potential ASPs:

ALGORITHM 3.3 (Identify one of the ASPs). Without loss of generality, let's assume that $x_s \le x_t$, and $y_s \le y_t$, (Case 1 in Algorithm 3.2).

• Step 1: Run Algorithm 3.2 twice to determine the ARPs of S and T. Let Set_{ASP} represent the set of PEs (cells) who are ARPs for both S and T. If T is not an ARP of S, then **exit** the procedure by reporting "No *ASP*(S, T) exists."

• Step 2: [Check neighbors] PEs communicate with their neighbors to determine whether they are in SET_{ASP}.

• Step 3: Each PE in Set_{ASP} enables input ports N_i and W_i and one of the output ports, E_o or S_o, such that the enabled port connects a PE in the Set_{ASP}. If both output ports are eligible, one of them is randomly chosen. Note that for each PE in the Set_{ASP} (except T), there will be at least one such neighbor. As a result, the possible switch settings for a PE in Set_{ASP} are $\{N_i, W_i, E_o\}$ or $\{N_i, W_i, S_o\}$. T's predecessors will only have T as their successor.

• Step 4: S broadcasts a signal s on the subnet, the PEs that receive s form a unique ASP.

In conjunction with Algorithm 3.3, the following theorem can be proved.

THEOREM 3.1. *In a maze of dimension $N \times N$, if there exists one or more absolute shortest paths (ASPs) between $S(x_s, y_s)$ and $T(x_t, y_t)$, then the cells along one of the ASPs can be identified in $O(1)$ time on an RMESH of dimension $N \times N$.*

Proof. This theorem follows from two facts: (i) the existence of an ASP can be tested in O(1) time, and (ii) Algorithm 3.3 is correct and runs in O(1) time. We already proved (i). To prove (ii), we need to show that Algorithm 3.3 identifies a unique path between S and T with length equal to *Manhattan*(S, T) in O(1) time. By Proposition 2.2, the cells in $\text{Set}_{\text{ASP}}$are all inside the S–T rectangle. According to Step 3, S connects to a single successor and so do the processors in Set_{ASP} and only eastbound ("→"s) and southbound ("↓"s) connections are allowed. Since this path will never exit the S–T rectangle, if we start from processor S and follow the enabled switches, we are bound to arrive processor T after *Manhattan*(S, T) steps. This will be the same path formed by the processors receiving S's signal in Step 4 of the algorithm. Clearly, the path formed after Step 4 is unique. To prove this fact let's first assume the opposite, i.e., there exists more than one path. In such a case, there exists a processor in the path that connects to two successors, and this contradicts with the switch settings explained in Step 3 of the algorithm. Therefore, the path found is unique and its length is *Manhattan*(S, T). Obviously, the running time of Algorithm 3.3 is O(1). ■

3.2. Shortest Duplex-Path (SDP)

As the maze gets congested due to blockages, finding an ASP between two given terminals gets more difficult (see Table I). In this case, we might look for a shortest path that is not an ASP. In general, any path can be thought of as a combination of one or more ASPs. A special case occurs when the path is composed of two ASPs (see Fig. 1c). This kind of a path is called a duplex-path and can be formally defined as follows:

TABLE I
Percentage of Paths vs Percent of Blockages on a Maze of Size 40 × 40

Blockage%	ASP	DASP	PolyPath	ASP + DASP
10	76.6	20.4	3.0	97.0
20	52.2	32.6	15.2	84.8
30	43.4	27.6	29.0	71.0
40	33.6	29.0	37.4	62.6

DEFINITION 3.2. A path between two cells S and T is called a *duplex-path* (*DP*) if it contains a cell $c(i, j)$ such that c divides the path into exactly two absolute shortest paths, $ASP(S, c)$ and $ASP(c, T)$. The cell $c(i, j)$, an ARP of both S and T, is called an *articulation point* (*AP*). The shortest path among all DPs is called *shortest duplex-path* (*SDP*). If a SDP(S, T) is the shortest path available between S and T, then it is called *duplex absolute shortest path* (*DASP*).

Figure 1c shows a duplex-path between S and T, $DP(S, T)$ which is also a $DASP(S, T)$. Clearly, the length of $DP(S, T)$ (denoted by $|DP(S, T)|$) is ($Manhattan(S, c) + Manhattan(c, T)$). With the above definition, any ASP is a special DP. It can be easily proved that if $|DP(S, T)| > Manhattan(S, T)$, then the articulation point (AP) c must lie outside the S–T rectangle. Let's consider an articulation point c whose shortest distance to S–T rectangle is i as shown in Fig. 3. It is shown in the figure that any duplex-path $DP(S, T)$ of which c is an AP

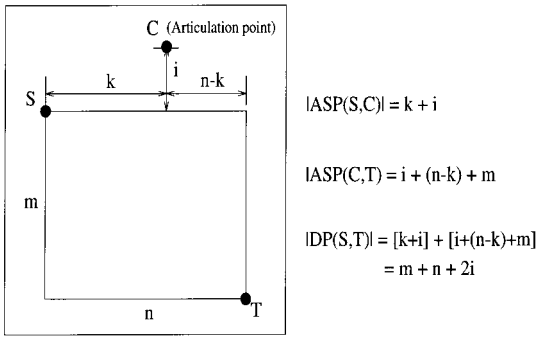

FIG. 3. Calculation of $|DP(S, T)|$.

satisfies the relation $|DP(S, T)| = m + n + 2i$ where m and n are the height and width of the S–T rectangle. Now, we give the definitions of DP^i (Fig. 4) and APC^i (Fig. 5).

DEFINITION 3.3. Given a source cell $S = (x_s, y_s)$ and a target cell $T = (x_t, y_t)$ on a maze, assuming that $|x_s - x_t| = m$ and $|y_s - y_t| = n$, $DP^i = \{DP(S, T) | |DP(S, T)| = m + n + 2i\}$.

DEFINITION 3.4. Consider a completely unblocked maze. A contour formed by APs associated with the paths in DP^i is called APC^i (ith articulation point contour). Special subnets formed along APCs are called APC-buses.

ALGORITHM 3.4 (FIND_SDP: Find the shortest duplex-path). This algorithm assumes that PEs know the coordinates of S and T and they have marked all the articulation points

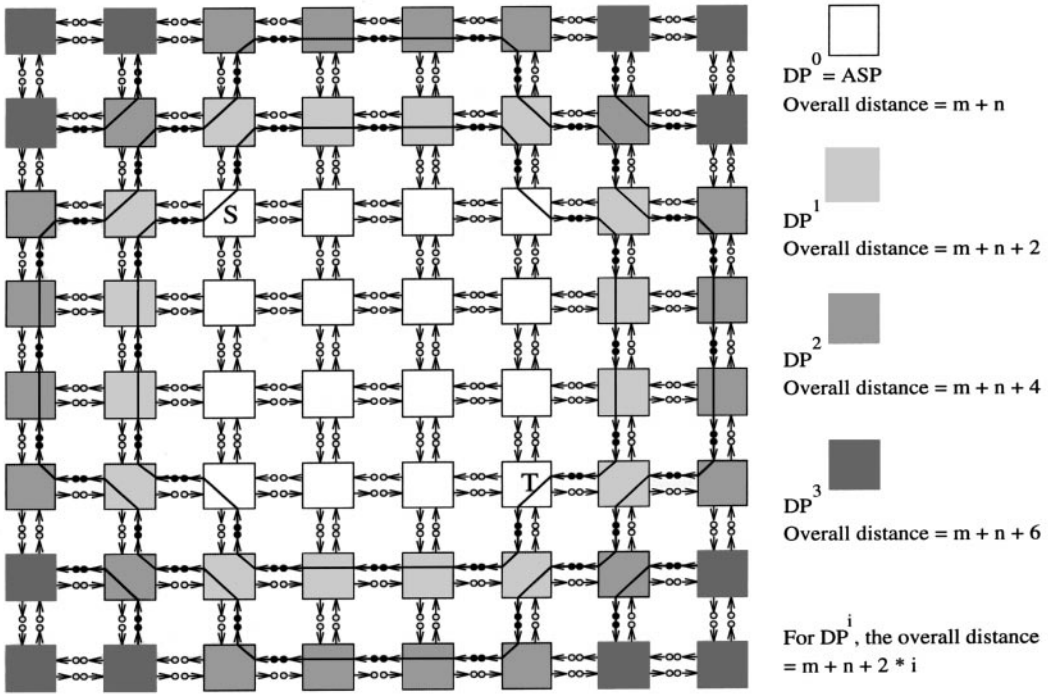

FIG. 4. Formation of octagonal APC-buses.

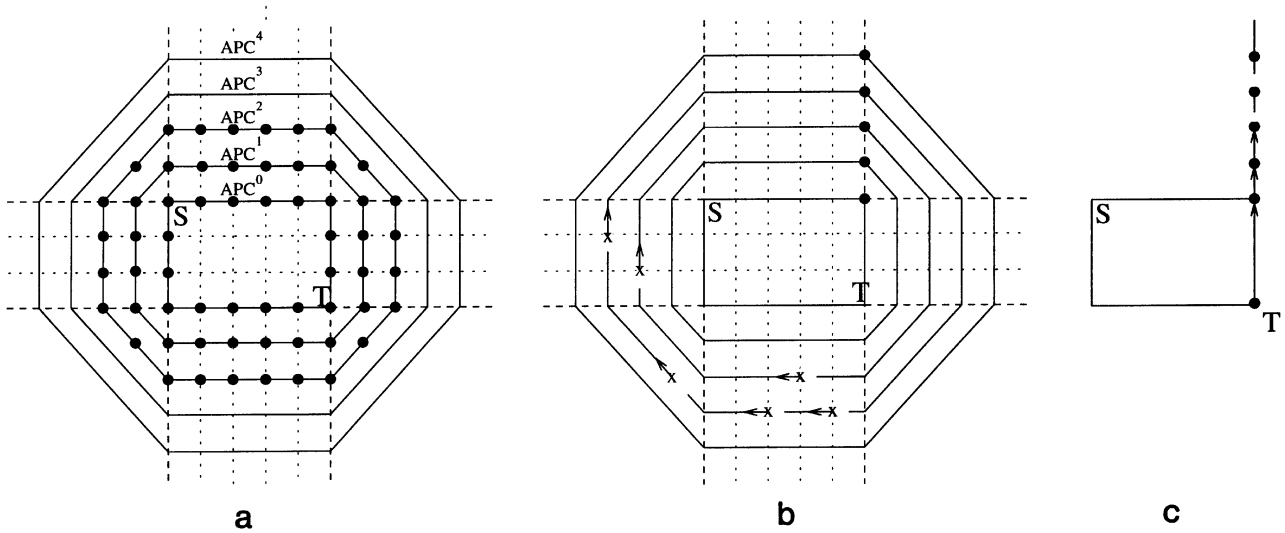

FIG. 5. (a) AP-contours and (b, c) two network configurations for Algorithm 3.4. (b) x, articulation point; •, APC-master. (c) APC-masters form a special column bus.

with respect to S and T using Algorithm 3.2. Without loss of generality, let's assume that $x_s \le x_t$ and $y_s \le y_t$.

• Step 1: [Form APC buses] PEs connect their input/output ports in such a way that they form buses as shown in Fig. 5b. These buses are called APC-buses. If a PE is an AP, it disables its input port and enables only one of its output ports in the clockwise direction as shown in Fig. 5b. If a PE is not an AP, it enables both input and output ports in the clockwise direction as to allow the signal to propagate along the contour. Each APC^i designates one PE as the APC^i-master, the processor on the top right corner of the APC^i whose column number is the same as T's (see Fig. 5b). Note that each PE's decision for which ports to enable/disable can be determined uniquely based on its relative position with respect to S and T, and whether it is an AP or not. Since PEs cannot make diagonal connections, they will connect to the diagonal PE through another neighbor as shown in Fig. 4. In this configuration, corner PEs will have two buses (contours) passing through it. In fact, these corner PEs belong to only one APC^i, but in order to form octagonal APC-buses, a second bus also needs to pass through them. They only participate in one read/write operation in any cycle.

• Step 2: [APs send their coordinates] All the AP(s) write their coordinates on the bus and APC-masters read the bus. Because of the way APC-buses are set up, there will be no conflict on the bus, and APC-masters will receive either no signal or the coordinates of the closest AP (in the counterclockwise direction). Let's assume that if APC^i has APs, then APC^i-master records $(x_{AP^*}, y_{AP^*})^i$ as the coordinates of the closest AP (AP*).

• Step 3: [APC-masters form a special column bus] APC-masters form a column bus which pass through T as shown in Fig. 5c. Those APC-masters which received no signal in Step 2 and the PEs along the east-edge of the S–T rectangle enable both $\{S_i, N_o\}$ switches while the others connect only their $\{S_i\}$ switch. With this configuration, only the closest APC-master will be connected to the same bus with T.

• Step 4: [Find the closest APC^i] T broadcasts a special signal t on the column bus setup in Step 3, and APC-masters read the bus. The APC-master which receives t identifies itself as the closest APC-master. Let's assume that kth APC-master wins.

• Step 5: The winning APC^k-master broadcasts the coordinates of its own AP*, $(x_{AP^*}, y_{AP^*})^k$ and all other PEs read the bus.

• Step 6: Run Algorithm 3.3 twice to mark the PEs along two ASPs; $ASP(S, AP^{*k})$ and $ASP(AP^{*k}, T)$.

4. SINGLE SHORTEST PATH ALGORITHM (SSPA)

As the maze gets congested, the number of available paths between two terminal points will decrease and the likelihood of finding an ASP or a DASP will become smaller. Let's now consider an extreme case: There is one and only one path available between S and T and this path is not a member of ASP or DASP class. Can we design an efficient RMESH algorithm to find the cells on this unique path? Single shortest path algorithm (SSPA) given below can find such a path if the reachability network for S forms a tree (called an *H-tree* [12]) as shown in Fig. 6. If it is not a tree, the algorithm can recognize the situation after at most $(2 \log N + 1)$ steps. In [12], the worst-case running time of the SSPA algorithm is proved to be O(log N). Assuming that the reachability network is a tree, the algorithm starts from the leaf cells and progressively eliminates all of the cells not belonging to the main path (solid line in Fig. 6a). For this purpose, the cells are classified into several types:

Terminal cell: exactly one of its neighbors is on the H-tree.
Path cell: exactly two of its neighbors are on the H-tree.

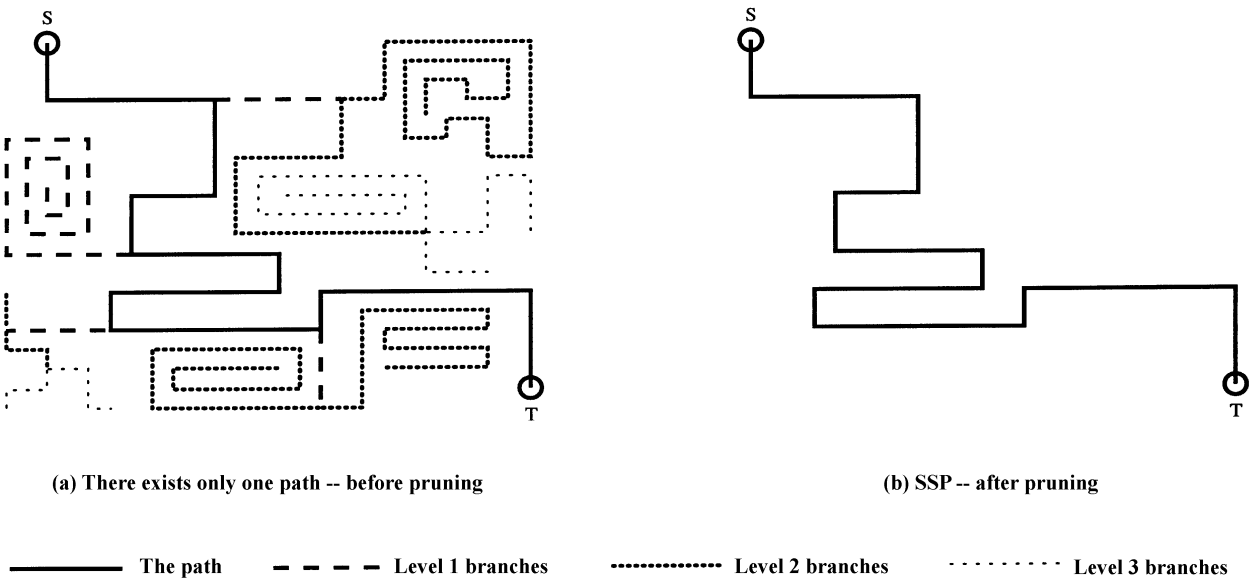

FIG. 6. A sample SSP and its H-tree (height $k = 3$).

Fork cell: more than two of its neighbors are on the H-tree.

ALGORITHM 4.1 (SSPA Algorithm). This algorithm finds the only shortest path available between S and T on an $N \times N$ RMESH by iteratively pruning the unnecessary branches of the H-tree.

- Step 1: [Mark the cells in the reachability set of S] Every nonblocked cell sets all of its switches. S broadcasts a special signal s and all the nonblocked cells read. If a cell receives s, it marks itself as "active"; i.e., it is reachable from S.
- Step 2: [Cells determine their types] Each active cell determines if it is a *terminal*, *path*, or a *fork* type cell. This can be completed in a total of 8 steps by communicating with 4 neighbors, each taking 2 steps (odd–even, even–odd communications).
- Step 3: [Prune those branches whose ends are marked with a fork and a terminal cell] *Path* and *terminal* cells connect to their active neighbors. S, T, and the *fork* cells disable all of their I/O ports. The terminal cells (excluding S and T) broadcast a special signal on the bus, and all active cells read the bus. Those cells receiving the signal mark themselves as "nonactive."
- Step 4: Repeat steps 2 and 3 until the termination condition is satisfied.

Termination Condition: The algorithm terminates when there are no more fork cells left in the tree or after looping $(2 \log N + 1)$ times (refer to [12] for a proof). To check for the first condition, after Step 2, fork cells disable all of their i/o ports, and S broadcasts a signal s on the bus. If T receives s, then there are no more fork cells left in the tree, and termination is reached.

In [12], it is proved that, if the reachability network of S is an H-tree, the SSPA Algorithm should prune the tree completely and find the SSP in at most O(log N) steps. If the reachability network of S contains cycles, then there will be fork cells left in the network even after $(2 \log N + 1)$ steps. If this happens, the algorithm terminates by concluding that the reachability network for S contains cycles.

5. SIMULATION RESULTS

To demonstrate the usefulness of the algorithms presented, experiments were conducted on a sequential machine to simulate the process of finding the shortest paths on a maze with blockages. Initially, a maze of size 40×40 with a certain amount of blockages was constructed. To mimic the blockages that may occur in a real maze-routing process, blockages are formed by selecting random source and target cells and connecting them with shortest paths until the percentage of blockages reached a certain value, e.g., 10%. Then, the Lee's algorithm [8] was run repeatedly to find the shortest paths between 100 randomly selected source and target cells. Each time a path was found, it was analyzed and put into one of the following categories: ASP, DASP, and PolyPath. If a path did not fit the definition of ASP, or DASP, it was classified as PolyPath. To increase the confidence level in our experiments, five distinct runs were conducted for each blockage level. Experiments were repeated for the blockage levels of 20, 30, and 40%. The results are presented in Table I. Table I shows that if the amount of blockages is 20% or below, the likelihood of finding an ASP or a DASP is 84.8%. When 40% of the maze cells are blocked (which is reasonably high), only 62.6% of the time, the shortest path is one of ASP, or DASP. This result indicates that unless the maze is badly congested

with blockages (which usually does not occur until after many terminal pairs are connected), our algorithms will be successful in finding the shortest path with a high probability. Hence, we mention the practical importance of our algorithms while stating that a constant-time algorithm to find the shortest path in general is yet to be discovered.

6. CONCLUSIONS

The maze-routing problem is defined as finding the shortest path between a source and a destination point following the nonblocked cells in a maze. Because of a perfect match between the topology of a maze and a mesh and the fact that RMESH architectures offer efficiency and reconfigurability in interprocessor communications, these architectures are excellent candidates to solve the maze-routing problem efficiently. In this paper, we described several time-efficient maze-routing algorithms on RMESH of size $N \times N$. To the best of our knowledge, this is the first study which presents efficient algorithms to solve the maze-routing problem on an RMESH architecture. The constant-time RMESH algorithms presented include testing the existence of a path between two terminals, finding an absolute shortest path (ASP), and finding the shortest duplex path (SDP) on a 2D RMESH. One other time-efficient algorithm, SSPA, to find a uniquely available path is also given. Simulations are conducted to obtain statistical data about the distribution of the specific types of paths under varying amounts of blockages. The results indicate that a large percentage of the shortest paths that exist between two randomly selected terminals fall into one of the categories studied in this paper which confirms the practical significance of our algorithms.

For future research, it remains a challenge to find a shortest path (SP) or the length of a shortest path between two points in constant time considering an unrestricted path definition. Whether an O(1) algorithm for the general shortest path problem on an RMESH can be found is an open problem. We believe that the work presented in this paper will serve as an important stepping stone in that direction. We especially think that the SSP Algorithm has some promising implications toward an efficient solution for the maze-routing problem.

ACKNOWLEDGMENTS

The authors express their appreciation to the anonymous reviewers for their valuable comments.

REFERENCES

1. Li, H., and Maresca, M. Polymorphic-torus network. *IEEE Trans. Comput.* **38** (Sept. 1989), 1345–1351.
2. Box, B. Field programmable gate array based reconfigurable preprocessor. *Proceedings IEEE Workshop on FPGAs for Custom Computing Machine, April 1994*, pp. 40–48.
3. Ben-Asher, Y., Peleg, D., Ramaswami, R., and Schuster, A. The power of reconfiguration. *J. Parallel Distrib. Comput.* **13** (1991), 139–153.
4. ElGindy, H., and Wetherall, L. An l_1 voronoi diagram algorithm for the reconfigurable mesh. *Proc. of the 2nd Reconfigurable Arch. Workshop in IPPS'95, Santa Barbara*, April 1995.
5. Jenq, J. F., and Sahni, S. Reconfigurable mesh algorithms for fundamental data manipulation operations. In Ozguner, F., and Ercal, F. (Eds.). *Parallel Computing on Distributed Memory Multicomputers.* NATO ASI Series, Springer-Verlag, Berlin/New York, 1993, pp. 27–46. [Also available as Tech-Report 91-031 (Univ. of Florida) at ftp://ftp.cis.ufl.edu/cis/techreports/tr91/tr91-031.ps]
6. Miller, R., Prasanna-Kumar, V. K., Reisis, D., and Stout, Q. F. Parallel computations on reconfigurable meshes. *IEEE Trans. Comput.* **42**(6) (June 1993), 678–692.
7. Wang, B. F., and Chen, G. H. Constant time algorithms for the transitive closure and some related graph problems on processor arrays with reconfigurable bus systems. *IEEE Trans. Parallel Distrib. Systems* **1** (1990), 500–507.
8. Lee, C. Y. An algorithm for path connections and its applications. *IRE Trans. Electron. Comput.* (Sept. 1961), 346–365.
9. Aykanat, C., Kurc, T. M., and Ercal, F. Parallelization of Lee's routing algorithm on a hypercube multicomputer. *Proc. of 2nd European Dist. Mem. Comp. Conf. (EDMCC2), April 1991*, pp. 244–253.
10. Won, Y., and Sahni, S. Maze routing on a hypercube multiprocessor computer. *Proceedings of Intrl. Conf. on Parallel Processing, St. Charles*, Aug. 1987, pp. 630–637.
11. Yen, I.-L., Dubash, R. M., and Bastani, F. B. Strategies for mapping Lee's maze routing algorithm onto parallel architectures. *Proc. 7th Intrl. Parl. Proc. Symp., 1993*, pp. 672–629.
12. Ercal, F., and Lee, H. C. Fast algorithms for maze routing on an RMESH. Technical Report CSC 96-05, University of Missouri-Rolla, 1996. [Also available at http://www.cs.umr.edu/techreports/96-05.ps]

FIKRET ERCAL received the B.S. (with first-class honor) and M.S. in electrical engineering from Istanbul Technical University, and the Ph.D. in computer science from The Ohio State University in 1988. He was a scholar of the Turkish Scientific and Technical Research Council during 1971–1988. Between 1988 and 1990, he was an assistant and associate professor at Bilkent University, Turkey. He is currently an associate professor at University of Missouri–Rolla and the director of the computer vision laboratory. Dr. Ercal has published extensively in the areas of parallel algorithms, medical imaging, and computer vision and has co-edited a book. He served as the program committee member and chair for several international conferences and is the recipient of a faculty excellence award from McDonnell Douglas Co. in 1995. He is a senior member of IEEE and the Editor of IEEE TCPP Newsletter.

HSI-CHIEH LEE is an associate professor in the Department of Information Management, Yuan-Ze Institute of Technology, Taiwan. He received a B.S. in mathematics from National Taiwan University in 1989 and an M.S. and a Ph.D. in computer science from University of Missouri–Rolla in 1994 and 1996, respectively. Dr. Lee has published in the areas of parallel algorithms, medical imaging, and neural networks. He has worked as a network system engineer and R&D department leader in Meditech Corp., Tapei, between 1989 and 1991 and as a graduate research and teaching assistant during 1992–1996.

Received November 13, 1995; revised August 26, 1996, June 2, 1997; accepted June 4, 1997