A Parallel Adaptable Routing Algorithm and its Implementation on a Two-Dimensional Array Processor

TAKUMI WATANABE, MEMBER, IEEE, HITOSHI KITAZAWA, MEMBER, IEEE, AND YOSHI SUGIYAMA

***Abstract*—A new parallel-processing wire-routing algorithm is presented and implemented on a parallel processor. The two main features of the parallel algorithm are the control of the path quality and the finding of a quasi-minimum Steiner tree. Both Lee's maze algorithm and the proposed algorithm are implemented on an AAP-1 two-dimensional array processor, and the performance is compared to that of software programming on a general-purpose computer. It is shown experimentally that routing by the proposed algorithm implemented on the AAP-1 is 230 times faster than a software maze router run on a 1-MIPS computer for a three-pin/net circuit on a 256 × 256 grid.**

I. Introduction

With advances in VLSI technology, high-performance and high-density VLSI layouts have been required in the design automation field. Parallel processing has attracted a great deal of attention as a promising technique for satisfying these requirements. A number of special hardware engines have been proposed for design automation. The cellular array processor AAP-1 (Adaptive Array Processor) has been developed by NTT and has already been applied to two-dimensional data processing [1]–[3]. A placement problem has been implemented on the AAP-1, and it is shown that the processing time can be reduced to $1/0.06N$ (N = number of placement modules) compared to a software solution carried out by a sequential computer (1 MIPS) [4].

In the routing field, many sequential processing algorithms have been presented [5]–[8]; however, except for the Lee maze algorithm [5], parallel-processing algorithms have not been widely investigated. Iosupovici [9] has proposed a special hardware architecture for a maze routing. Adshead [10], [11] has reported an improvement over the conventional software approach by a factor of more than 60 by applying a distributed array processor (DAP) in an 8K gate array routing. Breuer and Shamsa [12] and Blank *et al.* [13] have also proposed a similar architecture for a maze routing. Nair *et al.* [14] and Hong *et al.* [15] have described the techniques for gate array routing in special hardware. In these hardware algorithms, the wave-expansion phase can be processed in parallel but the trace-back phase still remains sequential. A maze router will always find the shortest path for two-point nets, but no consideration has been given to the multipoint net problem.

Manuscript received February 22, 1985; revised October 28, 1986.
The authors are with Atsugi Electrical Communications Laboratories, NTT, 3-1 Morinosato, Wakamiya, Atsugi-shi, Kanagawa 243-01, Japan.
IEEE Log Number 8612848.

This paper presents a new parallel-routing algorithm suited for parallel processing and discusses its implementation on the AAP-1. The proposed algorithm, referred to as the parallel adaptable routing (PAR) algorithm, is based on the rectilinear expansion of zones by a given expansion distance. The PAR algorithm can be further subdivided into two algorithms: a controllable-path-quality algorithm and a quasi-minimum-Steiner-tree-finding algorithm. The former controls the path quality by changing the expansion distance. It is shown that this algorithm includes Lee's algorithm and parallel line-search or line-expansion algorithms at extreme expansion distances. For the latter algorithm, a quasi-minimum Steiner tree can be obtained by taking the local mutual relationship between the net terminals into account.

In Section II, the structure of the Adaptive Array Processor (AAP-1) is briefly outlined. The routing problem and the conventional Lee's maze algorithm are discribed in Section III. The controllable-path-quality algorithm is explained, and then the quasi-minimum-Steiner-tree-finding algorithm is described in Section IV. In Section V, implementations of the algorithm on the AAP-1 are discussed in addition to the Lee algorithm. Finally, experimental results are discussed in Section VI.

II. Overview of Adaptive Array Processor (AAP-1)

The AAP-1 is a high-performance two-dimensional SIMD cellular array processor which can operate as a back-end processor for a host computer. It contains 256 × 256 1-bit processing elements (PE's). The AAP-1 consists of five major components: a PE array, an array control unit, a data buffer memory, an instruction memory, and an interface unit, as shown in Fig. 1. The PE consists of a RALU and two data transfer units. The RALU has an ALU and two register files. Each PE can communicate with its nearest orthogonal or diagonal neighbor through the data transfer unit. The top/bottom and right/left PE's are connected to each other, resulting in what is called a torus structure.

An application program can be coded with Fortran and

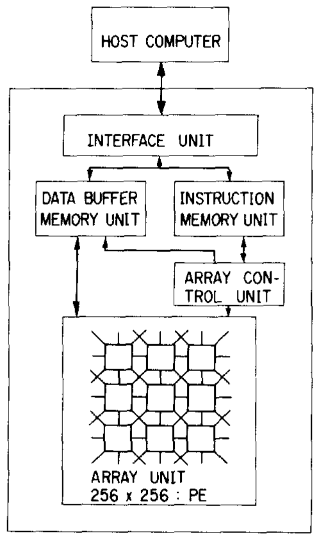


Fig. 1. AAP-1 system diagram.

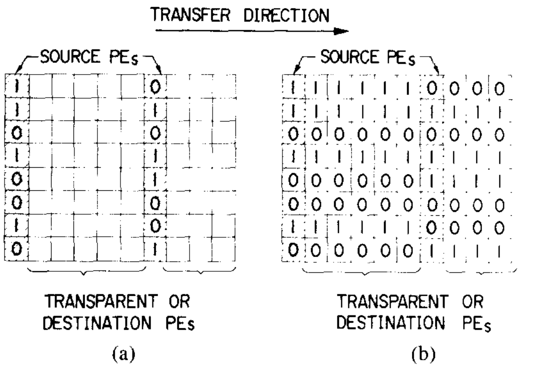

TRANSFER DIRECTION
SOURCE PEs
SOURCE PEs
TRANSPARENT OR DESTINATION PEs
TRANSPARENT OR DESTINATION PEs
(a)
(b)

Fig. 2. Ripple-through transfer (a) before execution and (b) after execution.

AAP-1 macroassembly language. The host computer generates AAP-1 machine codes, submits them to the AAP-1, and then receives its run results in the process of job execution. One of the most useful AAP-1 operations for parallel routing is the ripple-through operation shown in Fig. 2. Data are transferred to the distant PE's as far as the signal can propagate asynchronously within each clock interval. With this operation, the source PE's (arbitrarily defined) can transmit their own 1-bit datum along one of the eight directions (diagonal or orthogonal). The PE's other than the source act as the transfer PE's or destination PE's. Approximately 18 machine cycles are required if the data of the leftmost (rightmost) PE's are transferred to all PE's on their right hand (left hand). The situation is the same for a vertical data transfer.

III. Basic Routing Concepts

The specific routing problem encountered in this paper and the conventional Lee routing algorithm, which has usually been implemented on a two-dimensional array processor, are described in this section.

A. Routing Problem

Consider connecting points, which are treated as starting and destination points, and obstructions on a rectangular grid, and paths which are orthogonal lines passing through the grid. One connection can be made per grid cell. A routing area has two different layers available for routing. For ease of describing the algorithm, it will be explained as a one-layered problem. Fig. 3 shows an example of the routing problem.

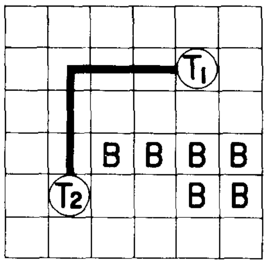


Fig. 3. Routing problem.

B. Conventional Lee Routing Algorithm

The Lee maze algorithm, which is a technique for finding the shortest path between two points, is composed of three phases: wavefront expansion, trace back, and cell clearing. The wavefront expands until the destination is hit; then a connecting path is located by following the descending wavefront labels back to the starting point. The wave-expansion and cell-clearing phase can be easily performed by parallel processing on a two-dimensional array processor. Thus, when compared with a sequential approach, the time complexity can be reduced from a square to a linear increase. A maze router will always find only a minimal-length path between two points, and no consideration has been given to a multipoint net problem.

IV. Parallel-Routing Algorithm

In the conventional routing method, a multipoint net problem is usually solved by decomposing the problem into two-point net subproblems. These subproblems are solved sequentially as the path found previously is used as the starting point. Depending on the order in which points are routed, this procedure sometimes results in the choice of redundant or lengthy paths. Fig. 4 shows an example of a multipoint net connection problem. In this figure, the solid lines show the multipoint net routing that a conventional maze router finds in some cases; the broken lines show the desirable result.

The parallel adaptable routing (PAR) algorithm consists of two categories: the controllable-path-quality algorithm (PAR-1) and the quasi-minimum-Steiner-tree-finding algorithm (PAR-2). PAR-1 controls the path quality, i.e., path length and number of corners, by changing the expansion distance. PAR-2 finds a quasi-minimum Steiner tree for a multipoint net. PAR-1 and then PAR-2 are explained in this section. The routing problem considered is the same as Lee's maze routing algorithm.

A. Controllable-Path-Quality Algorithm (PAR-1)

The Lee maze algorithm and the line-search algorithm have traditionally been treated as being different routing

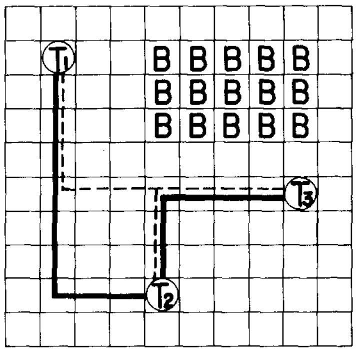


Fig. 4. Multipoint net routing problem. Solid lines denote typical results of conventional routers, broken lines the desirable results. T_1, T_2, T_3 are connecting points, and B denotes obstructions.

techniques. Both algorithms can be expressed in PAR-1 by introducing the expansion distance D_{ex}. Additionally, different quality paths can be obtained by choosing different D_{ex} values. Path quality entails both path length and the number of path corners or vias. This algorithm differs from the original Lee algorithm in that the expansion distance is not limited to unity but can take any value D_{ex} for each step. When D_{ex} is set to unity, it is equivalent to the original Lee algorithm. When D_{ex} is set to infinity, it functions as the parallel line-search or line-expansion algorithm. Fig. 5 shows three typical expansion diagrams when D_{ex} is unity, 3, and infinity, respectively. An example of two-point net routing for D_{ex} = 4 is given in Fig. 6. The algorithm procedure is as follows.

A. Path-Search Phase

Step 1. (Initialization)
First, clear all cells. Set obstruction flag (BC), start flag (SC), and target flag (TC) cells. Set SC to expanding cells (EC). Set label number or search number $NL = 1$.

Step 2. Expand zones from the EC vertically and horizontally to the distance D_{ex} unless this expansion is obstructed. Then, set LABEL = NL on the expanded zone (Fig. 6(a)–(c)).

Step 3. If one of the expanded zones hits the target cell (TC) then go to trace-back phase (Fig. 6(c)).

Step 4. Set $NL = NL + 1$.

Step 5. Set the expanded zone and starting cells to the EC.

Step 6. Go to step 2.

B. Trace-Back Phase

Step 1. (Initialization)
Set label number $LX = NL$. Set TC to a corner or terminal cell C_{LX}. Set SC to C_0. If $LX = 1$ then go to step 3.

Step 2. Expand the line in the same manner as in the path-search phase from the C_{LX}. One of the expanded lines hits a cell labeled $LX - 1$ (C_{LX-1}) (Fig. 6(d), (e)). Then store a line segment between C_{LX} and C_{LX-1} as a part of the path.

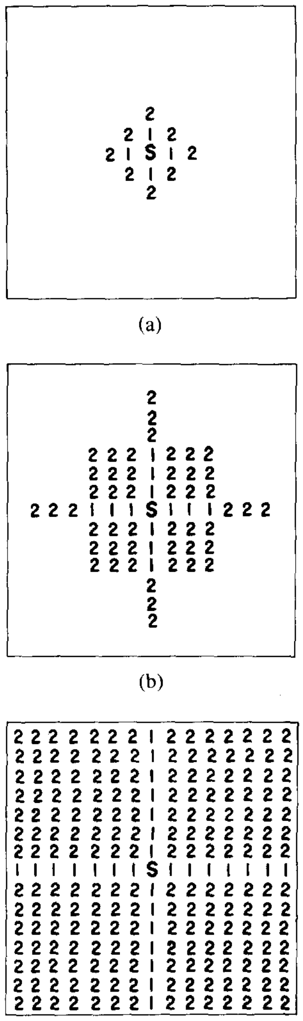


Fig. 5. PAR-1 expansion diagram. (a) D_{ex} = 1 (wave expansion). (b) D_{ex} = 3. (c) D_{ex} = ∞.

Step 3. If the C_{LX-1} is the SC then go to step 6 (Fig. 6(f)).

Step 4. Set $LX = LX - 1$

Step 5. Go to step 2.

Step 6. For the next net, reset the previous path as the BC. Then go to the path-search phase.

The PAR-1 finds a path with the minimum number of search operations (N_s). Fig. 7 shows that various paths can be obtained between two points by changing the D_{ex} value. When D_{ex} is set to unity, it finds a minimal-length path (P_l). When D_{ex} is set to infinity, or L_a, where L_a is greater than the length of the routing area, the PAR-1 finds a minimum corner path (P_c) since the number of the corners is equal to $N_s - 1$. For a two-layered problem, where horizontal routings are isolated from vertical routings and connections between them are made through via holes, it finds a minimum via path. If D_{ex} is set low, the total path length is shortened and the number of vias is increased. Conversely, if D_{ex} is set large, the total path length becomes long and the number of vias decreases for multinet routing problems. Assume that three different paths exist

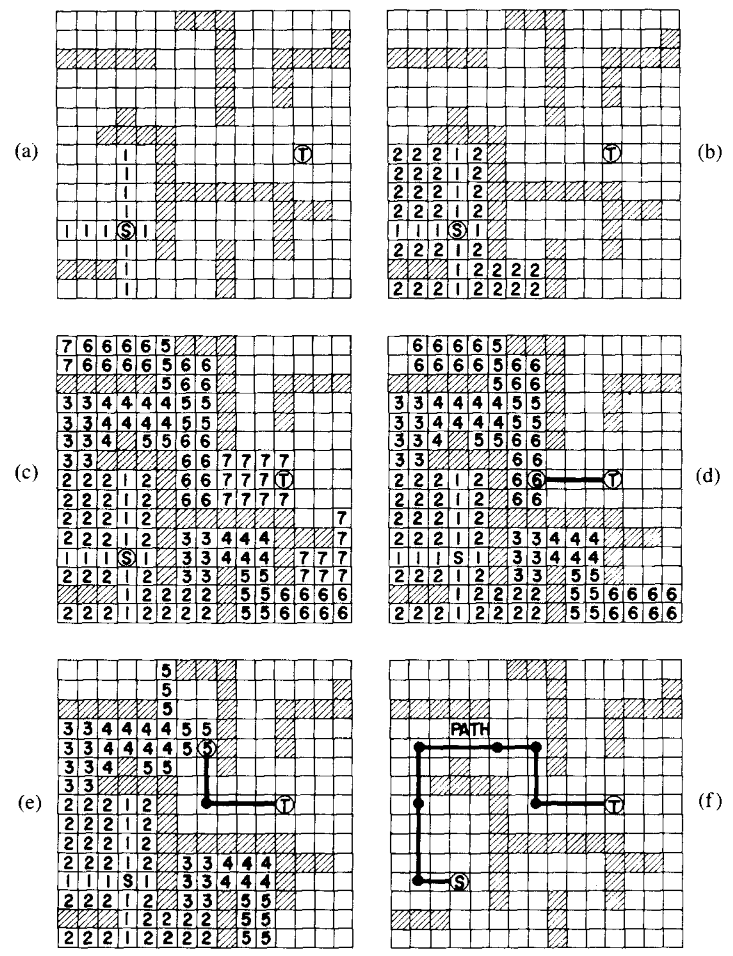


Fig. 6. PAR-1 algorithm ($D_{ex} = 4$). Shaded areas represent obstacles. S is the starting cell and T the target cell. (a) Initial expansion. (b) Second expansion. (c) End of path-search phase. (d) First trace-back. (e) Second trace-back. (f) Path determination.

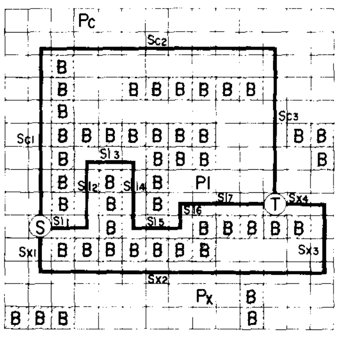


Fig. 7. Difference in paths with D_{ex} between two points (S, T). B denotes obstructions and P_c is the minimum corner path ($D_{ex} = \infty$). P_l is the minimum length path ($D_{ex} = 1$), and P_x is an alternate path ($D_{ex} = 4$).

between two points: P_c is a minimum corner path, P_l is a minimal length path, and P_x is an additional path. The total path length (L_c, L_l, L_x) of three paths can be calculated as follows:

$$L_c = S_{c1} + S_{c2} + S_{c3} + \cdots + S_{ci}$$

$$L_l = S_{l1} + S_{l2} + S_{l3} + \cdots + S_{lj}$$

$$(L_l \leq L_x \leq L_c,\ i \leq k \leq j)$$

$$L_x = S_{x1} + S_{x2} + S_{x3} + \cdots + S_{xk}$$

where S is the length of a path segment. The search numbers (N_{sc}, N_{sl} and N_{sx}) of the paths (P_c, P_l and P_x) are shown as follows:

$$N_{sc} = \lceil S_{c1}/D_{ex} \rceil + \lceil S_{c2}/D_{ex} \rceil + \cdots + \lceil S_{ci}/D_{ex} \rceil$$

$$= i \qquad (D_{ex} = \infty)$$

$$N_{sl} = \lceil S_{l1}/D_{ex} \rceil + \lceil S_{l2}/D_{ex} \rceil + \cdots + \lceil S_{lj}/D_{ex} \rceil$$

$$= S_{l1} + S_{l2} + \cdots + S_{lj} \qquad (D_{ex} = 1)$$

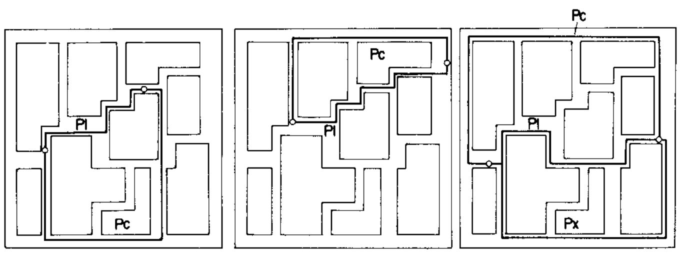
$$N_{sx} = \lceil S_{x1}/D_{ex} \rceil + \lceil S_{x2}/D_{ex} \rceil + \cdots \lceil S_{xk}/D_{ex} \rceil$$

$$(1 < D_{ex} < \infty)$$

where $\lceil x \rceil$ indicates a minimum positive number which is not smaller than the value x. Table I gives the search numbers (N_s) of Fig. 7 when the D_{ex} of P_c, P_l and P_x is set at L_a, 1, and 4, respectively. The algorithm selects the smallest one for each case, i.e., the path P_l (P_x, P_c) is selected if D_{ex} is set to 1 (4, L_a). This table shows that path quality is controlled by D_{ex}. Other P_c, P_l, and P_x examples are shown in Fig. 8. In this figure, when D_{ex} is infinity (L_a), the algorithm selects a minimum corner path even though the path selected is a detoured path (Fig. 8(a), P_c). When D_{ex} is unity, a minimal-length path is

TABLE I
SEARCH NUMBER

		Pc	Pl	Px
Number of corners		2	6	3
Ns	Dex = 1	25	17	19
	Dex = 4	7	7	6
	Dex = La	3	7	4



(a) (b) (c)

Fig. 8. Various paths with D_{ex}. P_c is the minimum corner path ($D_{ex} = \infty$); P_l is the minimum length path ($D_{ex} = 1$); and P_x is an alternate path ($1 < D_{ex} < \infty$). (a) Circuitous path (P_c). (b) Excess corner path (P_l). (c) An alternate path (P_x).

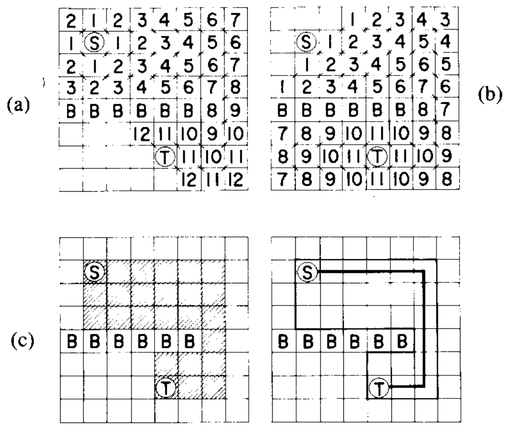


Fig. 9. Minimum corner with minimal-length paths. (a) Forward wave expansion. (b) Backward wave expansion. (c) restricted routing area. Shaded area is the union set of minimal-length paths.

selected even though it is an excess corner path (Fig. 8(b), P_l). Another path (Fig. 8(c), P_x) can also be obtained by setting D_{ex} to a suitable value ($1 < D_{ex} < L_a$). Thus, the path quality, path length, or number of path corners (vias) can be controlled according to net characteristics or terminal locations in the net.

This algorithm is applied to obtain a minimum-corner solution with a minimal-length path as follows. A wavefront with an ascending label is expanded from SC until it reaches TC; then a wavefront with a decreasing label is also expanded from TC to SC unless the area is obstructed as in Fig. 9 (double wave expansion). This is performed by using PAR-1 with $D_{ex} = 1$ (Fig. 9(a) and (b)). The area whose ascending and descending labels coincide is selected. This area is called a restricted routing area (Fig. 9(c)). A path is then found by using PAR-1 ($D_{ex} = \infty$) within the restricted routing area. This is a minimum-corner path with minimal length, because the restricted routing area is the union set of minimal-length paths in which PAR-1 ($D_{ex} = \infty$) finds a minimum-corner path (Fig. 9(d)).

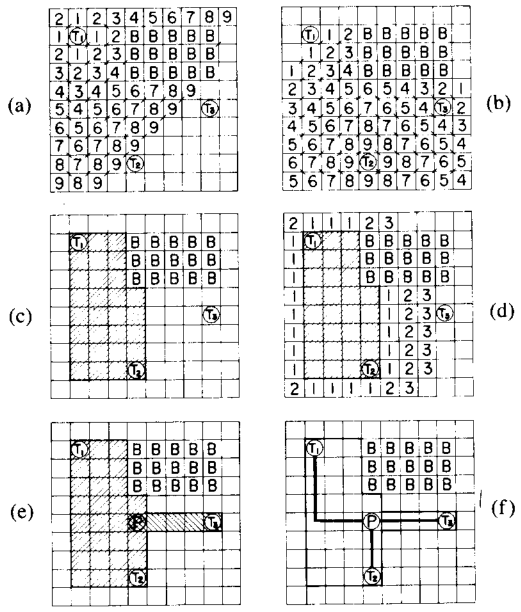


Fig. 10. Basic PAR-2 algorithm procedure. (a), (b) Double wave expansion between T_1 and T_2. (c) Restricted routing area 1 (right hatched area). (d) Double wave expansion between right hatched area and T_3. (e) Restricted routing area 2 (left hatched area). Common area of right and left hatched area is a branch point (P). (f) Path determination.

B. Quasi-Minimum-Steiner-Tree-Finding Algorithm (PAR-2)

A parallel-routing algorithm for a multipoint net (PAR-2) which solves the redundantly lengthy path problem is described below. Routing for a three-point net, which is the basic procedure for multipoint net routing, is first explained in Fig. 10. In this particular problem, the path between points T_1 and T_2, which is the nearest point from T_1, is found and then an attempt is made to connect point T_3 anywhere on a predetermined path. Whether the path between T_1 and T_2 is best or not depends on the position of the third point (T_3). Therefore, the restricted routing area (shaded area in Fig. 10(c)), which is obtained by the double wave expansion (Fig. 10(a) and (b)), is left without a specific path being determined until a branch point is found. The double wave expansion is made between this shaded area and T_3, and another restricted area is then obtained (left hatched area in Fig. 10(e)). A branch point P is appropriately selected by taking the logical AND between the two restricted routing areas, i.e., the left and the right hatched areas (Fig. 10(e)). The best path for $T_1 - T_2 - T_3$ which passes through the selected branch point P is thus found (Fig. 10(f)). In addition, the result does not depend on the order of the connecting points (T_1, T_2, T_3). The routing problem of a multipoint net can be solved by iteratively applying this three-point routing technique. The algorithm for multipoint routing is described as follows.

Step 1. (Initialization)
Set the number of connecting points in the net to NPIN, and set the area number $NL = 1$. Clear all cells. Set blocked cells. Then,

choose a starting cell (arbitrarily selected) t_{NL} and set it to expanding cells (EC).

Step 2. (Routing area restriction (double wave expansion)) Continue wave expansion ($D_{\mathrm{ex}} = 1$) of an ascending label from EC until one of the other connection points is hit. Then, treat the point so hit (t_{NL+1}) as a destination point (TC_{NL}). Continue wave expansion ($D_{\mathrm{ex}} = 1$) of a descending label from TC_{NL} until EC is hit. Then, obtain a restricted routing area Z_{NL} and give that area the label NL.

Step 3. If $NL = 1$ then go to step 6.

Step 4. Set a logical AND for EC and Z_{NL} as a branch point (P_{NL}). If P_{NL} is plural, then select one among them.

Step 5. If P_{NL} does not include any connection point in EC, then go to step 10.

Step 6. If $NL + 1$ is equal to NPIN, then go to step 13.

Step 7. Reset all labeled areas and previously determined path components to EC.

Step 8. Set $NL = NL + 1$

Step 9. Go to step 2.

Step 10. (Path determination) If P_{NL} is a part of a previously determined path component, then go to step 11; otherwise, relate the three points t_{NL-x}, t_{NL}, and P_{NL} to Z_{NL-x}, where NL and $NL - x$ are label numbers of the area in which P_{NL} is included ($1 \le x < NL$). Determine part of the path between the three points contained by Z_{NL-x} using the PAR-1 ($D_{\mathrm{ex}} = \infty$), where P_{NL} is to be a branch point. The routing area must be restricted in the labeled $NL - x$ area. Add the newly obtained path component to the previously determined path components. Then, eliminate Z_{NL-x}.

Step 11. If all points are connected with restricted areas, then go to step 13.

Step 12. Go to step 6.

Step 13. Determine the path by connecting the two points included in the remaining labeled areas using the PAR-1 ($D_{\mathrm{ex}} = \infty$).

A more definite procedure for a multipoint connection is explained using the example in Fig. 11. The restricted routing areas for t_1, t_2, t_3, and t_4 are obtained by the double wave expansion, and labels 1, 2, and 3 are then given to them (Fig. 11(a)-(c)). Label 3 does not contain any point other than the destination t_4 (Fig. 11(c)). The two areas labeled 2 and 3 intersect at point p_3. A path component is determined in the order of $p_3 - t_2$ and t_3. Then, eliminate the area labeled 2 from the restriction diagram (Fig. 11(d)). For the remaining part, the same process is repeated, and a path component $t_1 - p_4 - t_2$ is determined. When all points are connected with the restricted routing areas and path components, the remaining path components between $p_4 - t_5$ and $p_3 - t_4$ are determined (Fig. 11(e) and (f)).

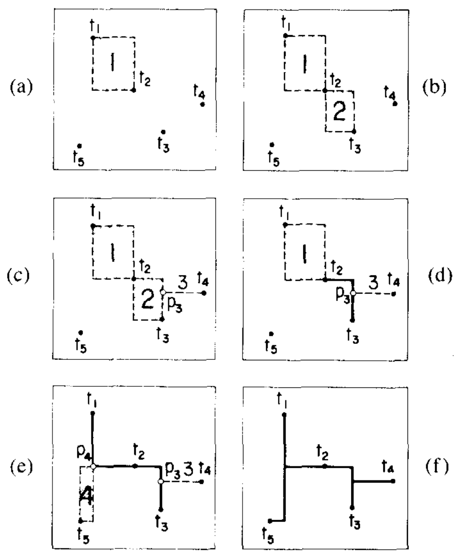


Fig. 11. PAR-2 algorithm. The area bounded by broken lines is the restricted routing area. (a) First routing restriction (t_1, t_2). (b) Second routing restriction (t_2, t_3). (c) Third routing restriction (t_4). P_1: branching point. (d) First path determination between t_2, t_3, and t_4. (e) Fourth routing restriction (t_5). P_2: branching point. (f) Final result.

V. Implementation on AAP-1

Lee's algorithm and the PAR algorithm have been implemented on the AAP-1. The grid cell array shown in Section III corresponds to the PE array of the AAP-1 shown in Fig. 12. The flags for obstructions and for starting and target (destination) cells are all set in the PE register. All of the starting and target cells, i.e., terminals, become obstructions unless they belong to the net concerned. For a multilayered problem, the cells in the same location in different layers are handled in the identical PE's. Then the expansion between the two layers is made by using an intra-PE data transfer operation. When the routing space size of the problem exceeds the physical array size (256 × 256), the routing can be accomplished by simply mapping or folding the space into several sheets of 256 × 256 (16). The net information is stored in the AAP-1 data buffer memory and sent to the PE array as required. The complete routing process is carried out in the AAP-1 without the need for communicating with the host computer. The routing results, i.e., line segment coordinates, are sent to the AAP-1 data buffer memory unit whenever a net connection is completed. When routings are finished for all nets, the AAP-1 sends the results back to the host computer. The data transmission time between the AAP-1 and the host computer is negligible for multinet routing. Fig. 13 shows the AAP-1 processing flow chart.

A. Parallel Lee Maze Router

1) Path-Search Phase: Wave expansion is performed by a shift operation (up, down, left and right directions), which is the conventional method of data transfer in array processors and intra-PE data transfer operations. Parallel processing is carried out in this phase. Therefore, the time required to expand from a starting cell (SC) to a target (TC) cell is proportional to the distance between them.

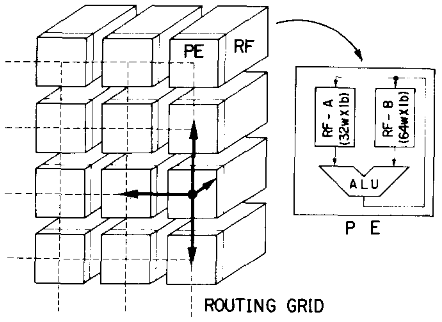


Fig. 12. Routing grid on PE array.

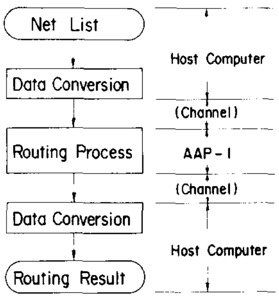


Fig. 13. AAP-1 processing flow chart.

2) Trace-Back Phase: Shift operations and intra-PE data transfer operations are also used in this phase. Descending labels on the cells are then traced back sequentially. Thus, the time required to trace back from TC to SC is also proportional to the distance between them.

B. PAR-1

1) Path-Search Phase: Wave expansion is realized by four directional shifts or ripple-through (RT) operations in parallel with an intra-PE data transfer operation. When, $D_{ex} < L_a$, where L_a is the length of the routing area size, i.e., the array size, the unit expansion of D_{ex} is made by shifting D_{ex} times. Therefore, the required time is proportional to the distance between SC and TC, just as in a parallel Lee's maze router.

When the $D_{ex} = L_a$, the RT operation accomplishes the path search extremely rapidly compared to the shift operation. Here, the starting and obstruction cells are to be source PE's. The RT operation rectilinearly propagates the datum "1" for the former and the datum "0" for the latter, respectively. The time required to reach SC from TC is proportional to the search number (N_s) but not to the distance between them.

2) Trace-Back Phase: This procedure is a sequential process in itself. However, the RT operation makes it possible to trace back each path segment rapidly. The operations required are as small as the number of path segments to be traced back from SC to TC. The time required to trace back from TC to SC is also proportional to the search number (N_s) or the number of path corners.

C. PAR-2

The routing area restriction (the double wave expansion) is made by using the expansion of PAR-1 ($D_{ex} =$

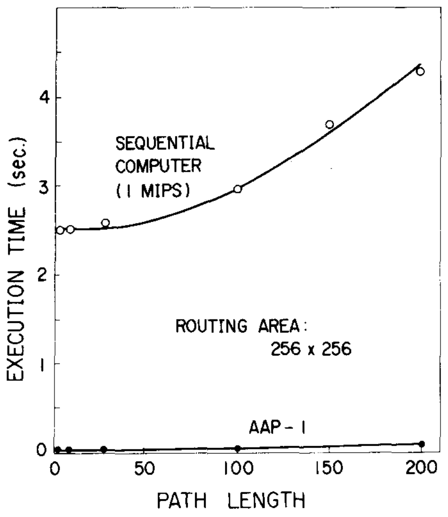


Fig. 14. Maze routing time on AAP-1 and 1-MIPS sequential computer.

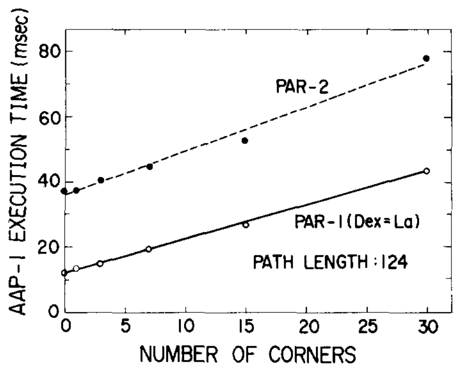


Fig. 15. AAP-1 execution time versus number of path corners on 256 × 256 grids.

1), i.e., wave expansion. The time required is proportional to the distance between SC and TC. For the path-determination phase, PAR-1 ($D_{ex} = L_a$) is applied. Thus, the path determination time is proportional to the number of corners in the path.

VI. Experimental Results

The experimental results for two-layered problems using the Lee algorithm and the PAR-1, PAR-2 implemented on the AAP-1 are described in this section. Fig. 14 shows a comparison of the execution time of a maze router implemented on an AAP-1 and on a 1-MIPS sequential computer. In this case, the AAP-1 can execute a maze-routing algorithm about 100 times faster than a 1-MIPS sequential computer.

The PAR-1 and PAR-2 algorithms are applied to various problems, and several fundamental characteristics of these routing algorithms have been obtained. The relationship between the number of path corners and the AAP-1 execution time is shown in Fig. 15. Fig. 16 shows the relationship between path length and execution time. The PAR-1 ($D_{ex} = L_a$) execution time is not proportional to the path length but rather to the number of path corners, since the path search is made by a ripple-through operation. On the other hand, the PAR-2 execution time (for

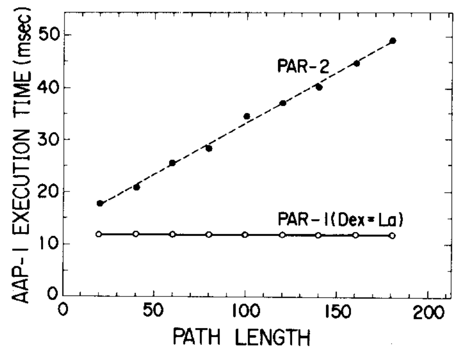


Fig. 16. AAP-1 execution time versus path length on 256 × 256 routing grids.

TABLE II
Time Complexity Summary

Routing Phases		PAR				MAZE	
		two - point		multi - point		AAP	Software
		AAP	Software	AAP	Software		
Array Init.		K	O (L²)	K	O (L²)	K	O (L²)
Restriction	FWD	—	—	O (R)	O (R)	—	—
	BWD	—	—	O (R)	O (R²)	—	—
Routing	Path Search	* O (R)	O (R²)	O (T)	O (R²)	O (R)	O (R²)
	Trace Back	O (T)	O (R)	O (T)	O (R)	O (R)	O (R)
Total		O(R) + O(T)	O(L²) + O(R²)	O(R) + O(T)	O(L²) + O(R²)	O (R)	O(L²) + O(R²)

L denotes (array size)$^{1/2}$; R is the routed path length; T is (number of turns) − 1; and K is a constant.
*$O(T)$ when $D_{ex} = L$.

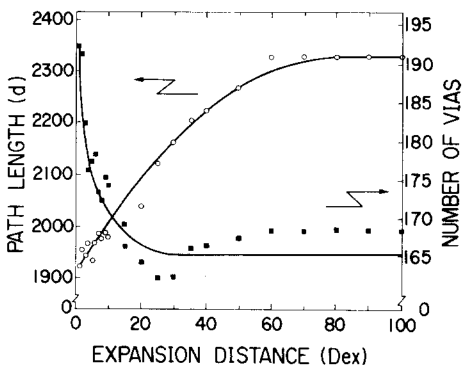


Fig. 17. Change of the path quality with D_{ex}.

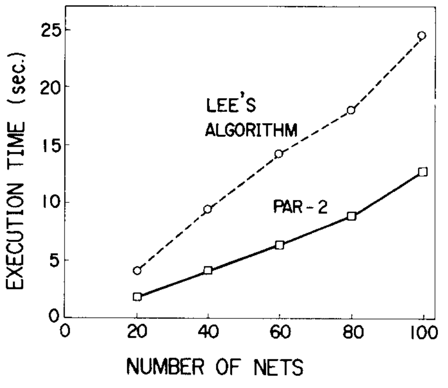


Fig. 18. Comparison of PAR-2 execution time with Lee's algorithm on AAP-1.

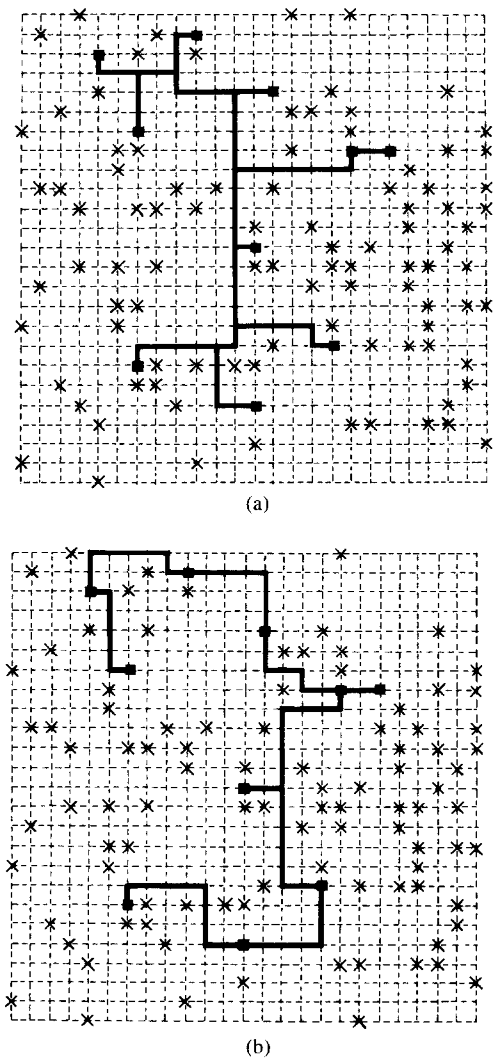


Fig. 19. Multipoint net routing. (a) PAR-2. (b) Conventional maze router. ■: connecting point. ×: obstruction.

multipoint connections) is proportional to both the number of corners and the path length because it uses a maze-type wave-expansion in the double-wave-expansion phase. Fig. 17 shows the changes in total path length and number of vias as a function of changes in D_{ex} (PAR-1) for a multinet problem. In this test problem is a two-layered problem where horizontal routings are isolated from vertical routings on 113 × 90 routing grids. Thus, a path corner corresponds to a via hole. Fig. 18 shows a comparison of the execution time between the PAR-2 and the maze router implemented on the AAP-1. The test problem uses random connection data for an average of three pins/net in a 256 × 256 routing area. The average execution time per net using the PAR-2 and the maze router were 100 and 230 ms, respectively. Although the PAR-2 procedure is more complicated than that of a simple maze router, the PAR-2 is faster than the maze router on the AAP-1. This is because the maze implementation on the AAP-1 requires many more sequential operations in its trace-back phase than does the PAR-2. Therefore, the AAP-1 can execute the PAR-2 algorithm about 230 times faster than a simple maze router in a 1-MIPS sequential computer.

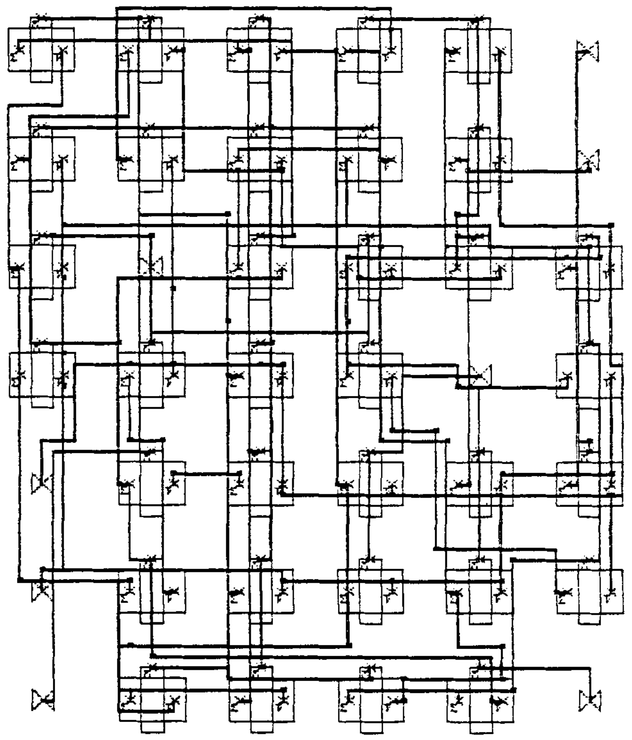


Fig. 20. PAR-2 routing result for a D-type flip-flop routing with 21 nets and 110 terminals on 113 × 90 grids. AAP-1 execution time is 1.95 s.

Table II summarizes the time complexities of each router. It can be seen that the processing time complexities for the proposed algorithm are linear, whereas the software sequential approach is not. Fig. 19 shows typical results obtained using a PAR-2 (Fig. 19(a)) and a conventional maze router (Fig. 19(b)) for multipoint routing on a 25 × 25 grid. The lengths of the two paths obtained were 57 and 64, and the numbers of corners were 17 and 20, respectively. Evaluating the path quality, an attempt was made to manually shorten the path length of the PAR-2 results having two to 14 connecting points per net. The connection data and obstruction data were given randomly, and routing was made in a 25 × 25 grid. In this test problem, the path length could be manually shortened in only six of 58 cases. The improvement ratios for the six cases were less than 3 percent. Fig. 20 shows lthe routing results obtained by PAR-2 on a D-type flip-flop circuit with 21 nets and 110 terminals. The AAP-1 execution time was 1.95 s.

VII. Conclusions

A new routing algorithm suited for parallel processing, the parallel adaptable routing (PAR) algorithm, has been described. The PAR-1 algorithm controls path quality by changing the expansion distance. It functions as a parallel line-search or line-expansion algorithm when the expansion distance is set at infinity and as Lee's algorithm when the expansion distance is set at unity. A PAR-2 algorithm for multipoint nets was also proposed which is effective for optimizing partial-path determinations. The PAR-2 can find a path having a quasi-minimum Steiner tree.

The Lee algorithm and the PAR algorithms have been implemented on an AAP-1 two-dimensional array processor. The run results show that the Lee maze routing algorithm on the AAP-1 can be executed 100 times faster than a sequential Fortran program on a 1-MIPS general-purpose computer, while the PAR algorithm implemented on the AAP-1 is 230 times faster for a three-pins/net circuit on 256 × 256 grids.

Acknowledgment

The authors would like to thank H. Mukai, T. Sudo, and T. Nakashima of the Integrated Electronics Development Division for their valuable advice and constant encouragement. The authors are also grateful to T. Kondo and T. Tsuchiya for their technical discussions and invaluable advice on the AAP-1 and to K. Ueda and H. Miyashita for their thoughtful comments.

References

[1] T. Sudo *et al.*, "An LSI adaptive array processor," in *ISSCC Dig. Tech. Papers*, Feb. 1982, pp. 122-123.
[2] T. Kondo *et al.*, "An LSI adaptive array processor," *IEEE J. Solid-State Circuits*, vol. SC-18, no. 2, Apr. 1983.
[3] T. Kondo *et al.*, "A large scale cellular array processor: AAP-1," in *Proc. Computer Science Conf. '85 ACM*, Mar. 1985.
[4] Y. Sugiyama and T. Watanabe, "Parallel processing of logic module placement," *Electron. Lett.*, vol. 20, no. 5, pp. 219-220, 1984.
[5] C. Lee, "An algorithm for path connections and its applications," *IRE Trans. Electron. Comput.*, pp. 346-365, Sept. 1961.
[6] D. Hightower, "A solution to line-routing problems on the continuous plane," in *Proc. 6th Design Automation Workshop* (Miami Beach, FA), June 1969, pp. 1-24.
[7] K. Mikami and K. Tabuchi, "A computer program for optimal routing of print circuit conductors," in *Proc. Information Process. '68*, 1969, pp. 1475-1478.
[8] W. Heyns *et al.*, "A line-expansion algorithm for the general routing problem with a guaranteed solution," in *Proc. 17th ACM/IEEE Design Automation Conf.*, 1980, pp. 243-249.
[9] A. Iosupovici, "Design of an interative array maze router," in *Proc. IEEE Int. Conf. Circuits Comput.*, Oct. 1980, pp. 908-911.
[10] H. G. Adshead, "Employing a distributed array processor in a dedicated gate array layout system," in *IEEE Int. Conf. Circuits Comput.*, Sept. 1982, pp. 411-414.
[11] H. G. Adshead, "Towards VLSI complexity: The DA algorithm scaling problem: Can special DA hardware help?" in *Proc. 19th ACM/IEEE Design Automation Conf.*, June 1982, pp. 339-344.
[12] M. A. Breur and K. Shamsa, "A hardware router," *J. Digital Systems*, vol. 4, no. 4, pp. 393-408, 1980.
[13] T. Blank *et al.*, "A parallel bit map processor architecture and algorithms for DA algorithms," in *Proc. 18th ACM/IEEE Design Automation Conf.*, June 1981, pp. 837-845.
[14] R. Nair *et al.*, "Global wiring on a wire-routing machine," in *Proc. 19th ACM/IEEE Design Automation Conf.*, June 1982, pp. 224-231.
[15] S. J. Hong and R. Nair, "Wire-routing machines—New tools for VLSI design," *Proc. IEEE*, vol. 71, pp. 57-65, Jan. 1983.
[16] T. Watanabe *et al.*, "A parallel maze router," unpublished, 1984.

*

Takumi Watanabe (M'86) was born in Fukuoka, Japan, on October 11, 1956. He received B.E. and M.E. degrees in electronic engineering from Kagoshima University, Kagoshima, Japan, in 1979 and 1981, respectively.

In 1981, he joined the Musashino Electrical Communication Laboratory, Nippon Telegraph and Telephone Public Corporation, Tokyo, Japan. He is now with Atsugi Electrical Communications Laboratories, NTT, Kanagawa, Japan. He has been engaged in the research and development of logic LSI design automation technologies. His research interests include parallel processing and computer-aided design, especially in VLSI layout.

Mr. Watanabe is a member of the Institute of Electronics and Communication Engineers of Japan and the Information Processing Society of Japan.

*

Hitoshi Kitazawa (M'85) was born in Nagano, Japan, on February 5, 1952. He received the B.S., M.S., and Ph.D. degrees in electronic engineering from the Tokyo Institute of Technology, Tokyo, Japan, in 1974, 1976, and 1979, respectively.

In 1979, he joined the Musashino Electrical Communication Laboratory, Nippon Telegraph and Telephone Public Corporation (NTT). He is now engaged in the research and development of logic LSI design automation technologies.

Dr. Kitazawa is a member of the Institute of Electronics and Communication Engineers of Japan and the Information Processing Society of Japan.

*

Yoshi Sugiyama was born in Shizuoka, Japan, on March 11, 1947. He received the B.S. degree in electrical engineering from Shizuoka University, Hamamatsu, Shizuoka, Japan, in 1969.

In 1969, he joined the Electrical Communication Laboratory, NTT, at Musashino. He is now engaged in research and development of logic LSI design automation technologies.

Mr. Sugiyama is a member of the Institute of Electronics and Communication Engineeers of Japan and the Information Processing Society of Japan.