A SIMPLE AND EFFECTIVE GREEDY MULTILAYER ROUTER FOR MCMS

Young-Jun Cha[1] *Chong S. Rim*[2,3,*] *Kazuo Nakajima*[3]

[1]Electronics and Telecommunications Research Institute, Taejon, Korea
yjcha@etri.re.kr
[2]Department of Computer Science, Sogang University, Seoul, Korea
csrim@ccs.sogang.ac.kr
[3]Electrical Engineering Department, University of Maryland, College Park, MD, 20742, USA
kazuo@eng.umd.edu

ABSTRACT

We present a very fast area router for MCM design. Given a set of nets and a routing area, it repeatedly routes as many nets as possible in two layers selected from top to bottom until all the nets are routed. The router uses a line sweep technique and a simple net selection algorithm. By using efficient data structures and a novel priority scheme, the routing can be finished in time almost proportional to the routing area. For the benchmark circuits, the proposed router generates comparable routing results faster by over an order of magnitude (18 times on the average) than the previously reported best router.

1. INTRODUCTION

In the traditional electronic system implementation, each chip is packaged separately and the chips are interconnected in a printed circuit board (PCB). With the Multi-Chip Module (MCM) technology, several chips are packaged into a single chip, which enables us to integrate more circuits in the same area, to simplify the interconnection in the PCB, and to reduce the interconnection delay. Thus, the MCM technology is well suited for very high-performance VLSI system implementation[2, 3, 8, 13, 14].

An MCM device consists of IC chip modules and a package body which forms a multilayer routing area. The modules are usually placed on the top layer of the routing area, and the remaining layers are used for routing signals, power and ground, heat conductance, and signal redistributions[3, 14]. The modules may have input and output pads, which are directly connected to the top layer of the MCM package, or they may have their own input and output pins, which are connected to the pads surrounding each module on the top layer of the package[2]. We may need to use distributed vias or pin redistribution layers to connect the pads to the signal routing layers, and hence a pin redistribution procedure may be applied before routing[5]. In this paper we assume that this problem has been resolved so that the terminals of every net are already located in the signal routing area.

In general, an MCM routing area consists of a much larger number of grid points than those on conventional PCBs and a large number of layers are needed for routing. Therefore, the routing problem for MCM is much more difficult to solve, and hence a very fast and effective routing method is ever needed. Major figures of merits used in MCM routing include the number of routing layers, the number of vias, and wire length. Their minimization would lead to higher performance, higher yield, and lower fabrication cost.

In the past several approaches have been proposed for MCM routing[4, 6, 7, 9, 10, 11, 13, 15]. One approach[4, 7, 13] extends the traditional maze routing technique[12], which is not suited to large problem sizes often encountered in MCM routing. Another approach[9, 15] uses layer assignment in order to overcome the size and net ordering problems. In this approach, a global routing of nets is first obtained and then each net is assigned to layers so that wires of different nets do not intersect each other. A third approach[6] applies topological routing to MCM.

Recently, two routers, called SLICE and V4R[10, 11], were developed and the V4R was shown to be the best performer. Both algorithms iterate the process of sweeping selected columns of a routing region in one direction. In each such process, as many nets as possible are routed on a single layer for SLICE and on double layers for V4R. Since a matching-based algorithm is used at each selected column, they require high execution times and may not be able to cope with large MCMs of the future. The SLICE is even worse since it keeps all the net routing information on the entire single layer because of its reliance on maze routing for the case of routing failure.

In this paper, we propose a very *S*imple but very *E*ffective *G*reedy *R*outing *A*lgorithm, called SEGRA. It generates routing results very comparable to those of V4R but runs faster by over an order of magnitude (average 18 X). Unlike SLICE and V4R, our algorithm routes nets by sweeping each and every column of the area and in two directions simultaneously. We create a sweep line each at the left and right boundaries of the routing area, and perform the routing by moving the two sweep lines toward the center. In each sweeping SEGRA tries to route as many nets as possible on two layers, and this process is repeated until all the nets are routed. When a sweep line arrives at a new column, vertical routing of each net is tried by applying a simple priority-based algorithm. Since we use a small number of priority levels and efficient data structures for column and terminal searching, the routing can be done in almost linear time with respect to the routing area.

* This author of the paper was supported in part by 94' Non Directed Research Fund and 95' Grant For Faculty Research Abroad, Korea Research Foundation.

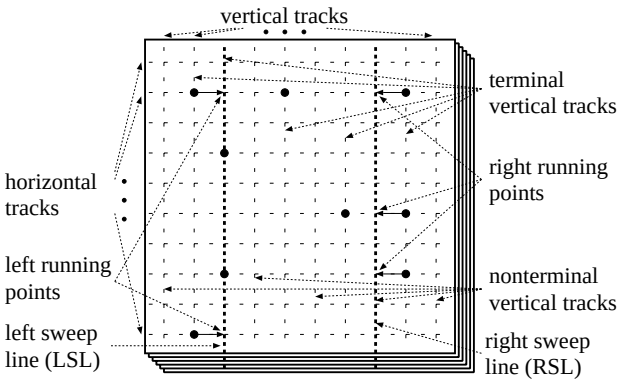

Fig. 1. An illustration for terminologies.

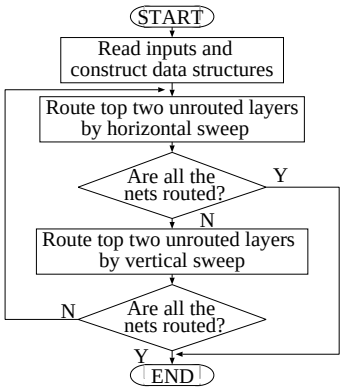

Fig. 2. Overall routing scheme.

Our router successfully completed the routing of the benchmark circuits used for the performance evaluation of SLICE and V4R. SEGRA runs about 90 times faster than SLICE, and about 18 times faster than V4R, on the average. For each circuit, SEGRA used the same number of layers as required by V4R, a smaller number of vias and a very similar total wire length.

Section 2 defines some basic terminologies and gives an overview of SEGRA. Section 3 describes the algorithm and Section 4 presents our data structures and analyzes the time complexity of our algorithm. The experimental results are shown in Section 5, followed by conclusions in Section 6.

2. TERMINOLOGIES AND ALGORITHM OVERVIEW

The inputs to our MCM routing problem are sets of modules and nets and a multilayer routing area. The outputs are sets of wire segments placed in each layer and vias created between specific adjacent layers. Our goal is to complete the routing so as to minimize the total length of wires, the number of vias, and the number of layers used.

We assume that the modules are placed on the top layer of the routing area and that any preprocessing such as pin redistribution has already been finished. Each layer in the routing area consists of a Manhattan style rectangular grid and each pin of every net is located on a grid point. We call the horizontal and vertical grid lines the *horizontal* and *vertical tracks*, respectively, which are depicted as lightly dotted vertical and horizontal lines in Fig. 1. We call the vertical tracks which have terminals *terminal vertical tracks* and the others *nonterminal vertical tracks*. Our router treats the two types of vertical tracks differently.

Our basic routing scheme is described in Fig. 2. We first read a netlist and the routing area information such as the numbers of horizontal and vertical tracks. While reading the netlist, if the net consists of more than two terminals (*i.e.*, it is a multi-terminal net), we obtain a minimum spanning tree[1] for the net and partition it into two terminal nets. Thus we assume that a net consists of two terminals throughout the paper. We denote the numbers of terminals, horizontal tracks, vertical tracks, and terminal vertical tracks by T, H, V, and V_t, respectively. We denote the number of horizontal tracks having terminals by H_t.

We construct an array for sweeping the routing region and another for searching terminal information. The former is a simple linear array of size proportional to $\max\{H, V\}$, called the *status array*. While sweeping the region, we use two status arrays for storage of information related to the sweeping. The array for terminals is designed to accommodate any necessary search such as finding on a specific horizontal track the nearest terminal from the current position. As explained in Section 4, its size is proportional to T and the operations on it can be done in logarithmic time.

While sweeping the routing area, SEGRA routes as many nets as possible in two adjacent layers currently selected from top to bottom, and the unrouted nets are considered for the next pair of layers. The sweeping direction is alternatively changed. If it is horizontal for the current two layers, the direction of sweeping for the next two layers is vertical, and vice versa (see Fig. 2). We use such a sweeping technique to improve the routing quality. In each process of sweeping, whether horizontal or vertical, we perform net routing in exactly the same way. Hence we describe our algorithm for a horizontal sweep only in the following.

Before the start of a sweep, we create two vertical lines called *left sweep line* (LSL) at the left end and *right sweep line* (RSL) at the right end of the routing area. The nets are then routed gradually as the two lines move toward the center of the area. If the two terminals of a net n are located on opposite sides of LSL and the routing is still going on, the routing for n is partially done on the left side of LSL. At this time, the routing to the right of LSL starts from the point on the line where the current partial wiring for n ends. We call this point the *left running point* of n. Similarly, if the terminals of n are located on opposite sides of RSL, the *right running point* of n is the point where RSL and the partial wiring result of n meet. See Fig. 1.

Following the movement strategy to be explained in the next section, LSL and RSL move to the right and the left, respectively. While this movement, when a sweep line meets a terminal of a net n, a running point is created for it. The running point moves with the sweep line until it can immediately be connected to the other terminal or its running point of the same net n. When such routing is done, it becomes part of the routing result of n which is the trajectory of the running points. We call the right running point (or the right terminal if it has not met RSL) the *target* of the left running point. Similarly, we call the left running point or the left terminal the target of the right running point.

Whenever a sweep line arrives at a new vertical track, the running points on the sweep line may move vertically toward their targets. The objective is to make a running point meet its target effectively. Similar vertical movements are made at selected columns in SLICE and V4R[10, 11]. However, their matching-based approach is likely to lead to longer running times. Our algorithm uses a greedy running point selection strategy for vertical movement. With a

```
horizontal_sweep ( H ) {
  Create LSL, RSL, and their status arrays;
  while ( LSL and RSL are not adjacent to each other
          and unrouted nets exist ) {
    repeat {
      if ( A terminal vertical track or RSL is just to the right
           of LSL, or all nets are routed )
        break;
      else Move LSL one grid point to the right;
      if ( LSL is on a terminal vertical track )
        Create running points on LSL, and update
        the status array;
      else
        Update the priority of each running point on LSL;
      vertical_move_on_LSL( 0, H − 1, 6 );
    }
    repeat {
      This part is the same as the above repeat block except
      for RSL and right instead of LSL and left. }
  } }
```

Fig. 3. Horizontal sweep procedure.

limited number of priorities used for the selection, SEGRA runs faster. We explain the priority scheme and the vertical movement algorithm in Sections 3.2 and 3.3, respectively.

3. ROUTING ALGORITHM

3.1. Horizontal Sweep

An outline of our horizontal sweep procedure is shown in Fig. 3. We assume that the horizontal and vertical wire segments are to be placed in odd and even numbered layers, respectively. Initially, the sweep lines LSL and RSL are located at just one grid point outside of the leftmost and rightmost vertical tracks, respectively. LSL first moves one grid point to the right and performs the operations described below. LSL continues making right moves over contiguous nonterminal vertical tracks and stops at the last such track before reaching a terminal vertical track. RSL then starts moving to the left in the same way as LSL, and this entire process is repeated until both lines are adjacent to each other or all the nets are routed. As explained below, this scheme does not require frequent updates of the status arrays, leading to a reduction in running time. The operations on LSL are now described for the two possible cases.

Case (i) LSL is on a terminal vertical track: We first determine the running points to be located on LSL. We create a left running point for each terminal located on the current vertical track. If another running point already exists at the same position, we remove this old running point and defer the routing of the corresponding net to a later trial. We keep all the other existing running points. If no running point exists at a grid point g on LSL and on the same horizontal track from g to the nearest terminal m to the right, we create a running point for m, provided that no running point is generated from m and point g and the position of m are within a predetermined distance. The running points created in this way are to allow for possible detours to improve routability.

The running points are stored in the status array for LSL. In addition, each entry of the array contains the closest terminal or running point located in the sweep direction and its distance from LSL. This information is used to detect an *obstacle*, which is a terminal or a running point that does not correspond to the net currently under consideration. The array also contains the information on the target of each running point. After running point adjustments, we update these information quickly using our data structures and binary search to be explained in Section 4.

Case (ii) LSL is located on a nonterminal vertical track: We keep the current running points. If no running point exists on a grid point on LSL, we create a running point for the nearest terminal to the right in the same way as when LSL is located on a terminal vertical track, and update a few more things in the status array.

After the creation of running points, we determine if a vertical movement is needed for each running point and if so, we move the point. We first find the priority according to the priority selection criteria to be described in Section 3.2. We explain the vertical movement of running points in Section 3.3. After completion of the vertical movement phase, we move LSL to the next right column and repeat the same procedure. However, if the next vertical track is a terminal vertical track, we temporarily suspend the movement of LSL, and starts moving RSL to the left.

3.2. Priority Selection

The priority of a running point measures how much vertical movement it needs. The running point with the highest priority which has not yet been tried is first chosen for such movement. There are six levels of priority, where 6 is the highest and 1 is the lowest. With this limited number of priority levels, our vertical movement algorithm runs very efficiently. The priority of each running point r_p is set by certain rules. They differ in the above two cases and are described below.

Case (i) LSL is located on a terminal vertical track:

1. If the horizontal track that accommodates r_p is empty entirely to the right of LSL or r_p and its nearest terminal or running point to the right on the same horizontal track correspond to the same net, we give priority 1 to r_p, since we can directly connect the two objects.
2. If there is an obstacle at the distance of four or more grid points to the right, we assume no urgency to move r_p vertically. Therefore, we give a lower priority than that determined by Rule 3 shown below. The exact priority setting is based on the following rule: If no running point has been created from the right side (i.e., on RSL), we give priority 2 to r_p since the routing problem for the net may be resolved with RSL. However, if a running point has been created on RSL, we give priority 3 to r_p since the problem is not resolved even on the right side.
3. If an obstacle is located within three grid points from the location of r_p to the right, we assume some urgency of moving r_p to another position. We give priority 5 if a running point is created on the right side and give priority 4 otherwise.

Case (ii) LSL is located on a nonterminal vertical track: We keep the priority determined on the previous terminal vertical track. However, if an obstacle is located within two grid points, we change the priority of r_p to 6. This is because r_p has only three trials for vertical movement before it is deferred to the next pass, and hence it needs to be routed as quickly as possible. Therefore, even if the sweep line is located on a terminal vertical track, there may be running points with priority 6. We try to move these points first during the vertical movement procedure to be explained below.

3.3. Vertical Movement

The procedure of vertical movement is given in Fig. 4, where U and L indicate the upper and lower limits of the search

```
vertical_move_LSL(L, U, p) {
  if (U ≤ L) return;
  do {
    Find an untried running point r_p whose priority is equal
     to p starting from U down to L;
    if (such an r_p is found)
      Mark r_p tried, and try to move r_p vertically;
  } while (r_p is moved or every running point has been
            tried)
  if (no running point has moved)
    if ( p > 2 ) vertical_move(U, L, p − 1);
    else return;
  else {
    Suppose that the vertical position of r_p is moved up
     from Y_d to Y_u.
    if ( p > 2 ) vertical_move(Y_u + 1, U, p − 1);
    vertical_move(L, Y_d − 1, p);
  }
}
```

Fig. 4. Vertical movement algorithm.

range, respectively, and p is the priority level currently under consideration. We start the vertical movement procedure with the initial settings of $p = 6$, $U = H - 1$ and $L = 0$, where H is the number of horizontal tracks. Note that the running points with priorities $> p$ are not considered for movement since they have failed at previous trials. If a running point is selected and successfully moved up from Y_d to Y_u, we apply the same procedure recursively to the intervals $[Y_u + 1, T]$ and $[B, Y_d - 1]$ on the same vertical track. Note that, in the interval $[Y_u + 1, U]$, we search for running points with priority $p - 1$ since the running points of priority p have already been tried in the interval.

Suppose that the procedure for vertical movement is called with parameters L, U, and p. We search the status array corresponding to the interval $[L, U]$ from top to bottom and find the first running point with priority p. If there is such a point, we try to move it vertically; else we call the procedure again with priority $p - 1$ if $p > 2$; otherwise we terminate the procedure.

Suppose that a running point r_p is found. We define the following position and sets of positions (see Fig. 5). Let t denote the vertical track on which LSL is placed, and assume that the target is located vertically higher than r_p.

1. p_b is the position at the intersection of the current sweep line and the horizontal track that holds the target.
2. S_c is the set of positions located on t between r_p and p_b but excluding r_p and p_b.
3. Let $e_m = 5 + \frac{1}{30}|S_c|$. Let S_u (respectively, S_l) be the set of e_m positions located just above p_b (respectively, below the position of r_p) on t. Note that the value of parameter e_m is determined experimentally and the number 5 is added for the case of a very small S_c.

The positions to which r_p tries to move are searched in the order of p_b, S_c, S_u and S_l. Note that S_u and S_l are extra margins for the case when the search fails at p_b and in S_c. If p_b or any other position in the above sets of positions is located outside the interval $[L, U]$, it is ignored. Also, if any obstacle is located within the interval, we reduce the search interval accordingly. The search procedure is described in detail below.

1. If the right side of the horizontal track from p_b is empty or the nearest terminal or running point from p_b to the right belongs to the same net as r_p, r_p is moved to p_b and routed immediately.

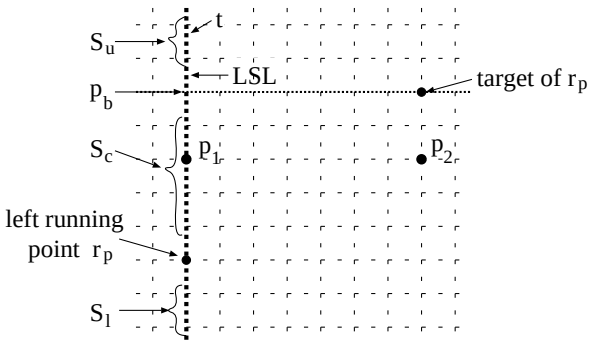

Fig. 5. Search range for vertical movement.

2. We search the points in S_c from top to bottom. If there is no obstacle on a horizontal track from a point p_1 in S_c to a point p_2 which is located on the same vertical track as the target (see Fig. 5), and p_2 and the target can be connected vertically, we move the running point to p_1. Checking the existence of an obstacle between p_1 and p_2 horizontally has already been done when we updated the status array. Obstacle checking between p_2 and the target can be done very quickly by performing the operation O2 to be described in Section 4. If we find an obstacle between p_2 and the target vertically, no more search in S_c is needed since we can not directly connect to the target from any point below the obstacle. Therefore, we do the vertical search only once.
3. If the search in S_c fails, we search the positions first in S_u from bottom to top and then in S_l from top to bottom.
4. If r_p has made a vertical movement before with Rules 2 and 3 above, only the first rule is applied. If it fails, no vertical movement will be made for r_p. This leads to a reduction of the number of vias.

If no good position for r_p to move to is found after completing the above procedure or if it has priority 6, we try to move the running point to the position where r_p will most likely be routed in the next movement. For this we search S_c, S_u and S_l as follows:

5. We start searching from the closest position to the target in the order of S_c, S_u and S_l. In this search, if the priority of r_p is changed to 1, we move r_p to the position. This leads to an easy connection between the two points at a later trial.

Note that position p_b is not searched again because we simply can not move r_p to the position. Also, if r_p has previously been tried for vertical movement by the same search procedure, it is not tried again. The same is true when it has moved by Rule 2 or 3.

Once the vertical movement is successful for the selected running point, we update the status array, adjust the search interval, and try to move a running point within the interval (see Fig. 4). If no running point with the same priority is movable, we search running points with the next lower priority from top to bottom.

4. ALGORITHM IMPLEMENTATION AND COMPLEXITY

We present an efficient implementation of the proposed algorithm and its complexity analysis. As described above, SEGRA frequently performs the following basic operations:

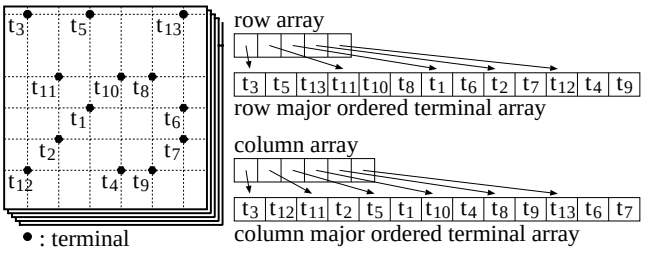

Fig. 6. Data structures for our routing algorithm.

Circuit	$V \times H$	S	κ
test1	300×300	98,878	1.10
test2	400×400	708,649	4.43
test3	500×500	1,452,580	5.81
mcc1	599×599	1,145,010	3.19
mcc2-75	$2,032 \times 2,032$	46,358,085	11.23
mcc2-45	$3,386 \times 3,386$	39,391,604	3.44

Table 1. The value of κ for each test circuit.

O1: Given a horizontal interval $[x_l, x_r]$ with $x_l < x_r$, and a horizontal track t_h, find the closest terminal to x_l on t_h in the interval if one exists.

O2: Given a vertical interval $[y_l, y_u]$ with $y_l < y_u$, and a terminal vertical track t_v, find the closest terminal to y_l on t_v in the interval if one exists.

In order to perform the above operations efficiently, we create two *terminal arrays*, one sorted by row major order and the other by column major order. They are used to access to a row and a column through the *column* and *row arrays* which point to the starting terminal in each row and each column, respectively. For example, in Fig. 6, the information on the terminal located at the third column and the first row may be obtained by first finding the start index of the third column in the column array and then accessing to the information at the first row on the row major ordered terminal array. These arrays enable us to determine the location of a terminal and the existence of a terminal at a given position.

The operations O1 and O2 can be done by a modified binary search algorithm[1] after finding the terminal array through the row or column array. If T_{tr} terminals are located on a vertical or horizontal track, each operation needs $O(\log T_{tr})$ time.

As described in the preceding section, SEGRA has two main phases: (i) status array upgrade when the sweep line is located on a terminal vertical track, and (ii) vertical movement. The most time consuming work in the first phase is to find the closest terminal on a vertical track in the sweep direction whenever the sweep line meets the terminal vertical track. It takes $O(H \cdot \log(V_t))$ time using operation O1 to perform this work. With a total of $O(V_t)$ status array upgrades, this phase runs in $O(V_t H \log V_t)$ time.

The time complexity of vertical movement is analyzed as follows: With six priorities, we search the status array at most 6 times to select a running point on each vertical track. This takes place $O(H)$ times in total. The sweep line moves over V vertical positions and these moves are repeated $L/2$ times if L layers are used, and hence the router searches $O(LVH)$ running points in total.

Suppose that a total of w running points are actually tried for their vertical movements. For each such point we search the positions in $S_u \cup \{p_b\} \cup S_c \cup S_l$ for possible vertical movements (see Fig. 5). Let S be the total accumulated number of such positions for the w running points. Testing the vertical movability needs $O(H_t)$ time since the existence of an obstacle in the horizontal direction is already done when updating the status arrays and operation O2 is applied at most twice for each running point. Therefore, the total time needed is $O(S + w \log H_t)$.

With the above analysis, the total routing time is $O(T_{tr} H \log T_{tr} + LVH + S + w \log T_{tr})$, where $T_{tr} \leq \max\{V_t, H_t\}$. Practically speaking, we may assume that the number of terminals is much smaller than the size of the routing area, which is HV. Therefore, we may assume that w is much smaller than HV since T is smaller than HV. Since $\log T_{tr}$ is a very small number compared with V and H, we can say that the time is approximated to $O(LVH + S)$. The vertical displacement between two running points is not so large, and hence the number of points for testing vertical movements may not be large. That is, S may be a few times larger than VH. Therefore, if we assume that $S = \kappa VH$, the total running time becomes $O((L + \kappa)VH)$. Table 1 shows the values of κ for the circuits we tested in this study. Clearly, κ is much smaller than V and H. Since the number L of layers used is also a very small number compared to V and H, the time complexity of SEGRA is approximately equivalent to that of searching the entire routing area several times.

The proposed algorithm use the status arrays, the row and column arrays, and the terminal arrays. Therefore, the space complexity of SEGRA is $O(\max\{H, V\} + T)$, which is similar to that of V4R[11].

Circuit	No. of Chips	No. of Nets	No. of Pins	Grid Size
test1	4	500	1,000	300×300
test2	9	956	1,912	400×400
test3	9	1,254	2,508	500×500
mcc1	6	802	2,495	599×599
mcc2-75	37	7,118	14,659	$2,032 \times 2,032$
mcc2-45	37	7,118	14,659	$3,386 \times 3,386$

Table 2. Features of benchmark circuits (pitch of mcc2-45: $45\mu m$, others: $75\mu m$).

Circuit	SEGRA	V4R	SLICE	3D–Maze
test1	4	4	5	4
test2	4	4	6	4
test3	4	4	6	4
mcc1	4	4	5	5
mcc2-75	6	6	7	–
mcc2-45	4	4	–	–

Table 3. The number of layers used for routing.

5. EXPERIMENTAL RESULTS

We have implemented the proposed algorithm in C. For the performance evaluation of SEGRA, we used the six benchmark circuits provided by Khoo and Cong[10, 11], who presented the routing results of V4R[11], SLICE[10] and 3D-Maze[11]. Table 2 shows the main features of the circuits[11], where test1, test2 and test3 are circuits obtained by randomly created two terminal nets, and mcc1, mcc2-75 and mcc2-45 are actual MCM circuits from MCC. Note that mcc2-75 and mcc2-45 are created with a grid pitch of 75 μm and 45 μm, respectively, from the same circuit.

Table 3 shows the number of layers used for routing by the four routers. Note that the entry marked as '-' means that the router failed to complete the routing. For each

Circuit	SEGRA	V4R	SLICE	3D–Maze
test1	2,080	2,250	2,013	2,975
test2	4,476	4,493	5,271	7,127
test3	5,833	5,855	6,892	9,347
mcc1	6,773	6,993	6,386	8,794
mcc2-75	36,362	36,438	47,864	–
mcc2-45	34,305	36,473	–	–

Table 4. The number of vias used for routing.

Circuit	SEGRA	V4R	SLICE	3D–Maze
test1	103,682	104,128	109,092	107,908
test2	270,021	271,067	286,723	273,642
test3	433,610	435,466	459,046	441,552
mcc1	433,202	394,272	402,258	397,221
mcc2-75	5,517,133	5,559,479	5,902,818	–
mcc2-45	9,135,340	9,130,705	–	–

Table 5. The total length of wires used for routing.

circuit tested SEGRA used the same number of layers as that by V4R which is the most recently reported router.

Table 4 gives the number of vias used for routing. We counted the number of vias as the number of connections between two wire segments on different layers where no other wire segment is involved in the middle of the connection. If the layers are not adjacent, the resultant connection leads to a so-called stacked via. We assume that the modules and I/O pins are located on the top and bottom layers, respectively. This via counting method is apparently used in V4R for the first three test circuits, which consist of two terminal nets only. As shown in the table, SEGRA used a smaller number of vias than V4R for each of the three circuits. If V4R used the same counting method, SEGRA performed better than V4R for the remaining three circuits, too.

Table 5 shows the total length of wires produced by the routers for each circuit. SEGRA used a shorter length of wires than V4R for four of the six circuits. Better results for the remaining two circuits would possibly be generated if we incorporate into SEGRA more efficient handling of multi-terminal nets, which is currently under investigation.

Table 6 shows the running time of each router. We ran SEGRA on a SPARC II compatible workstation with 32 Mbytes of memory in order to make a fair comparison with V4R. We have verified the fairness of this computing environment by actually running SLICE on the same computer. The resulting run times are very similar to those shown in Table 6. On the average, SEGRA ran about 90 times faster than SLICE. As V4R is about 5 times faster than SLICE, SEGRA is about 18 times faster than V4R for the circuits we tested.

Circuit	SEGRA	V4R	SLICE	3D–Maze
test1	0:00:02	0:01:00	0:02:00	0:08:00
test2	0:00:05	0:01:00	0:06:00	0:48:00
test3	0:00:08	0:03:00	0:12:00	1:40:00
mcc1	0:00:10	0:03:00	0:12:00	0:59:00
mcc2–75	0:02:59	1:06:00	8:15:00	–
mcc2–45	0:04:09	1:37:00	–	–

Table 6.Routing times (hr:min:sec).

6. CONCLUSIONS

We presented a very fast multilayer MCM router, called SEGRA. Further improvements of greedy routers such as SEGRA are often possible. However, the matter of fact is that SEGRA produced good routing results for the benchmark circuits 18 times faster on the average than the previously best performer V4R. The routing results are comparable to and sometimes better than those generated by V4R. The faster speed and the effectiveness of SEGRA mainly come from (i) a greedy method for net selection and vertical wire movement and (ii) efficient data structures.

7. ACKNOWLEDGMENT

The authors would like to express their gratitude to Prof. J. Cong of UCLA for supplying the benchmark data.

REFERENCES

[1] A. V. Aho, J. E. Hopcroft and J. D. Ullman, *Data Structures and Algorithms*, Addison-Wesley, Menlo Park, CA, 1987.

[2] H. B. Bakoglu, *Circuits, Interconnections, and Packaging for VLSI*, Addison-Wesley, Menlo Park, CA, 1990.

[3] A. J. Blodgett, "Microelectronic packaging," *Scientific American*, pp. 76–86, July 1983.

[4] J. D. Cho, S. Liao, S. Raje and M. Sarrafzadeh, "M^2R: Multilayer Routing Algorithm for High-Performance MCMs," *IEEE Trans. on Circuit and System I*, pp. 253–265, Apr. 1994.

[5] J. D. Cho and M. Sarrafzadeh, "The Pin Redistribution Problem in Multi-Chip Modules," *Proc. IEEE Application Specific IC Conference 91*, pp. P9-2.1, 1991.

[6] W. W-M. Dai, T. Dayan and David Staepelaere, "Topological Routing in SURF: Generating a Rubber-Band Sketch," *ACM/IEEE 28th Design Automation Conference*, pp. 39–44., June 1991.

[7] A. Hanafusa, Y. Yamashita and M. Yasuda, "Three-Dimensional Routing for Multilayer Ceramic Printed Circuit Boards," *Proc. IEEE Int'l Conf. on Computer-Aided Design*, pp. 386–389, Nov. 1990.

[8] D. Herrell, "Multichip module technology at MCC," *Proc. IEEE Int. Symp. on Circuits and Systems*, pp. 2099–2103, May 1990.

[9] J. M. Ho, M. Sarrafzadeh, G. Vijayan, and C. K. Wong, "Layer Assignment for Multichip Modules," *IEEE Trans. on Computer-Aided Design*, Vol.9, pp. 1272–1277, Dec. 1990.

[10] K. Y. Khoo and J. Cong, "A Fast Multilayer General Area Router for MCM Designs," *IEEE Trans. on Circuits and Systems II*, pp. 841–851, Nov. 1992.

[11] K. Y. Khoo and J. Cong, "An Efficient Multilayer MCM Router Based on Four-Via Routing," *IEEE Trans. on Computer Aided Design*, Vol 14, pp. 1277–1290, Oct. 1995.

[12] C. Y. Lee, "An Algorithm for Path Connections and its Application," *IRE Trans. on Electronic Computers*, Vol. EC-10, pp. 346–365, 1961.

[13] R. Miracky, T. Bishop, C. Galanakis, H. Hashemi, T. Hirsch, S. Madere, H. Muller, T. Ruswick, L. Simth, S. Sommerfeldt and B. Weighler, "Technology for Rapid Prototyping of Multi-Chip Modules," *Proc. IEEE Int'l Conf. on Computer Design*, 1991, pp. 588–591.

[14] H. B. Richard, P. M. William, and H. Jackson, "MultiChip Modules," *Proc. 26th ACM/IEEE Design Automation Conf.*, 1989, pp. 389–393.

[15] M. Sriram, and S. M. Kang, "Detailed Layer Assignment for MCM Routing," *Proc. IEEE Int'l Conf. on Computer-Aided Design*, 1992, pp. 386–389.