Computer Science Department Technical Report
University of California
Los Angeles, CA 90024-1596

A FAST MULTILAYER GENERAL AREA ROUTER FOR MCM AND DENSE PCB DESIGNS

K.-Y. Khoo
J. Cong

March 1992
CSD-920011

A Fast Multilayer General Area Router for MCM Designs

Kei-Yong Khoo and Jason Cong
Department of Computer Science
University of California, Los Angeles, CA 90024

Abstract

In order to reduce interconnection delay and increase packaging density, the multichip module (MCM) technology is used in the design of high-performance VLSI systems. A commonly used method for multilayer MCM designs is the three-dimensional (3D) maze routing, which suffers from a number of problems: it is very sensitive to the net ordering, it requires long computation time, and it often results in a large number of vias in the routing solutions. The objective of this research is to develop an efficient multilayer general area router as an alternative to the 3D maze router for solving the multilayer MCM routing problem. Our router, named SLICE, is independent of net ordering, requires much shorter computation time, and uses fewer vias. A key step in our router is to compute a maximum non-crossing bipartite matching, which is solved optimally in $O(n \log n)$ time where n is the number of possible connections. We tested our router on a number of examples, including two MCM designs from MCC. The total wirelength used by SLICE is only a few percent away from the optimal on average. Compared with a 3D maze router, SLICE is six times faster and uses 29% fewer vias. Another feature of SLICE is that it works on only a "thin slice" of a two-layer routing grid at a time, while a 3D maze router works on the entire three dimensional routing grid. Therefore, SLICE can successfully produce solutions for large MCM routing examples where 3D maze routers fail due to insufficient memory.

1. Introduction

As VLSI fabrication technology advances, interconnection and packaging (P/I) technologies have become a bottleneck in system performance [1,2,3]. In the traditional approach, each chip is first packed into single chip packaging (SCP) and then mounted on a printed circuit board (PCB). The area of each SCP is usually several times larger than the corresponding bare chip. As a result, the packing density is severely limited. Moreover, there exist two levels of inter-chip interconnections: the connections on SCPs and the connections on the PCB. The wasted space and the addition level of interconnections limit the packing density and degrade the system performance.

The multichip module (MCM) technology reduces the wasted space on a board and eliminates a level of interconnection by assembling and connecting bare chips on a common substrate. The substrate consists of multiple routing layers used for inter-chip interconnections. Without individual packaging for the chips, the bare chips can be placed much closer on the MCM substrate, which leads to a significant increase in packing density and decrease in interconnection delay.

Due to the high packing density in MCM designs, the MCM routing problem is more difficult than the conventional IC or PCB routing problems. First, MCMs may have far more interconnection layers than ICs. For example, the multi-chip module developed for the IBM 3081 mainframe has 33 layers of molybdenum conductors (including 1 bonding layer, 5 distribution layers, 16 interconnection layers, 8 voltage reference layers, and 3 power distribution layers [4,5]). Fujitsu's latest supercomputer, the VP-2000, uses a ceramic substrate with over 50 interconnection layers [6]. Moreover, unlike routing in ICs where the entire routing region can be naturally decomposed into channels and switchboxes, there is no natural routing hierarchy in MCM routing. The MCM routing problem is an immense three-dimensional general area routing problem where connection can be carried out almost everywhere in the entire multilayer substrate. Finally, the pitch spacing is much smaller and the routing result is much denser in MCM routing as compared to those in conventional PCB routing. Thus, traditional PCB routing tools are often inadequate in dealing with MCM designs[1].

Few methods are available for MCM routing. A commonly used method for multilayer MCM designs is the three-dimensional (3D) maze routing [6,7]. Although this method is conceptually simple to implement, it suffers from several problems. First, the quality of the maze routing solution is very sensitive to the ordering of the nets being routed, yet there is no effective algorithm for determining a good net ordering in general. Moreover, since each net is routed independently, global optimization is difficult and the final routing solution often uses a large number of vias despite the fact that there are many interconnection layers. Finally, 3D maze routing requires long computational time and large memory space. For example, one industrial example that we obtained from MCC has a 75 micron routing pitch and a routing area of $174 \times 174\ mm^2$; this results in a routing grid of 2032×2032 for a single layer! It is certainly not a trivial task to store such a grid for each layer and search in it efficiently.

Another method for multilayer MCM routing is to divide the routing layers into a number of $x-y$ layer pairs. Nets are first assigned to $x-y$ layer pairs and then two-layer routing is carried out for each $x-y$ layer pair (the x-layer runs horizontal wires and the y-layer runs vertical wires) [8]. Although this approach is efficient, it faces a few problems. First we have to pre-determine the number of the routing layers before we can carry out layer assignment. Moreover, the approach does not take advantage of the existence of the large number of routing layers. Thus, some nets may use many vias since they are forced to be routed within two layers. For high-performance MCM designs, vias not only increase the manufacture cost but also degrade the system performance since they form inductive and capacitive discontinuities and cause reflections when the interconnections have to be modeled as transmission lines [2].

Several efficient routers have been proposed for silicon-on-silicon based MCM technology [1,9,10,11]. Since the number of signal routing layers is usually small (2 to 4 layers) in this technology, some techniques for IC routing, such as hierarchical routing and rubber-band routing,

[1] Beside the problem of efficient utilization of routing resource, there are also several performance issues involved in MCM routing. For example, for high-performance designs, the wires need to be modeled as lossy transmission lines, where signal reflection and cross-talk need to be taken into consideration.

can be applied to yield good solutions.

The objective of our research is to develop an efficient multilayer general area router as an alternative to the commonly used 3D maze router for solving the multilayer MCM routing problem. Our router, named SLICE, has a number of advantages. First, it processes many nets simultaneously so that the routing solution is independent of net ordering. Moreover, it requires much shorter computation time and much smaller memory storage. Finally, it emphasizes planar routing so that most of the nets use very few vias. A key step in our method is to compute a maximum non-crossing bipartite matching, which is solved optimally in $O(n\log n)$ time (where n is the number of possible connections). We tested our router on a number of examples, including two MCM designs from MCC, and compared the results with those by a 3D maze router. On average, both routers use about the same total wirelength, but the 3D maze router is 6 times slower and uses 29% more vias. Another important feature is that SLICE works on only a "thin slice" of a two-layer routing grid at a time, while a 3D maze router works on the entire three dimensional routing grid. Therefore, SLICE can successfully produce solutions for large MCM routing examples where 3D maze routers fail due to the memory requirement.

The remainder of this paper is organized as follows. Section 2 formulates the multilayer MCM routing problem. In Section 3, we give an overview of our algorithm and we describe each step of the algorithm in detail. Experimental results and a comparative study are presented in Section 4. Finally, we discuss the extension of our work in Section 5.

2. Problem Formulation

The MCM routing problem consists of a set of modules, a set of nets, and a multilayer routing substrate. Modules (dies) are mounted on the top of the substrate by wire bonding, tape-automated bonding (TAB), or flip-chip bonding with solder bump connections. The substrate consists of multiple signal routing layers, with (possible) obstacles in some routing layers, such as power/ground connections and thermal conducting vias. The I/O terminals (pads) of the modules are connected to the substrate either directly or through routing to the external pads that surround the individual modules for engineering changes [2]. The pads are brought to the the first signal routing layer either directly through distribution vias or through one or more redistribution layers. The redistribution layers are required when the pads are too dense to be connected directly to the signal routing layers. A pin redistribution algorithm was presented in [12]. The goal of our MCM router is to complete the connections for the I/O terminals in each net using the signal routing layers in the substrate.

The signal routing layers in the substrate are numbered from top to bottom. We assume that there is a routing grid superimposed on each routing layer where the spacing between grid lines is determined by the routing pitch for the given P/I technology. We assume that the routing grid is a Manhattan grid. However, our algorithm can handle 45 degree routing as well. Two wires in adjacent signal routing layers can be connected by a via. Vias may be stacked on top of each other to connect wires in non-adjacent layers. Stacked vias can be formed in several ways, e.g., by filling the etched via with nickel in the AT&T AVP process or by plating copper posts as in

the MCC process [13]. Fig. 1 shows a cross section of a sample four layer routing region.

The output of the routing problem is a set of routing segments and vias that connect each net. The quality of the routing can be measured by the total wirelength, the number of vias, the number of wire bends (jogs) and the number of layers required to complete the routing. Long wire paths increase propagation time and should be avoided. Vias and wire bends degrades the signal's fidelity by introducing impedance discontinuity in signal paths thus should also be minimized. Vias usually cause more serious problems than jogs, so that our router gives via minimization a higher priority. Each additional routing layer increases the manufacturing cost and thus the number of layers should also be minimized.

3. Description of the algorithm

In this section, we present our fast multi-layer general area router, called SLICE, for MCM and single-sided PCB designs. We first give an overview of the entire algorithm, and then we describe each step in detail.

3.1. Overview of the algorithm

The basic idea behind our algorithm is to perform planar routing on a layer by layer basis. After routing on one layer, we propagate the terminals of the uncompleted nets to the next layer. Then we continue routing on the next layer and perform the single layer routing again. We repeat the process until all the nets are routed.

A crucial part of our algorithm is to compute a planar routing solution for each layer and try to connect as many nets as possible in that layer. For nets that cannot be completed in the layer, we try to perform a partial routing so that these nets can be completed in the next layer with shorter wires. We scan the routing region from left to right and process each pair of columns that has terminals at a time. For each adjacent column-pair, we compute a maximum weighted non-crossing matching (MWNCM) which consists of a set of non-crossing edges that extend from the left column to the right column. This gives us a topological planar routing solution between the column-pair. Next, we generate the physical routing between the current column-pair based on the selected edges in the non-crossing matching. Then, we move on to the next column-pair and

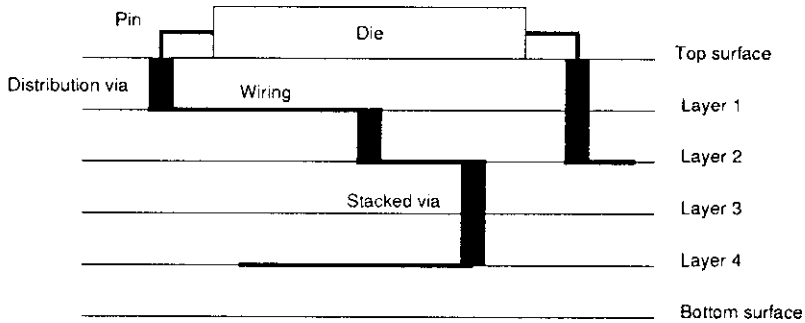

Fig. 1. Cross section of a multilayer (4 layer) routing region.

compute the non-crossing matching and physical routing again for that column-pair. The planar routing process is completed when the right end of the current layer is reached. Then, we distribute the terminals of the uncompleted nets so that they can be propagated to the next layer without causing local congestions. Since the left to right scanning operation in the planar routing results in mainly horizontal wires in the planar routing solution, in order to complete the routing in the vertical direction, we use a restricted two-layer maze router which is much faster than a general maze router to route within a thin vertical slice of the substrate one at a time from left to right[2]. We clean up the routing solution by removing unnecessary jogs and wires in the current layer. For nets that are not completed, their terminals are propagated to the next layer. Finally, we rotate the routing region by 90 degrees so that the scanning direction in the next layer is orthogonal to the one used in the current layer. The entire process is iterated until all the nets are routed. This top level algorithm is summarized in Fig. 2. We shall describe each step in detail in the remaining subsections.

3.2. Planar routing

The terminals that lie on the same vertical grid line form a *column*. In our planar routing algorithm, we scan the routing region across from left to right and perform routing between each column-pair. Let x_l and x_r be the x-coordinates of the left and right column of the current column-pair, respectively. Conceptually, a column-pair forms a channel and we define the channel capacity to be $C_{cap} = x_r - x_l$. During planar routing in the current layer, the terminals of the uncompleted nets are put in the list P_{prop}. These terminals will be propagated to the next routing layer.

Algorithm SLICE
  N = {list of nets};
  l = 1; /* Start at layer 1 */
  while ($N \neq \emptyset$) **do**
    Compute planar routing on layer l;
    Redistribute the uncompleted terminals on layer l;
    Perform restricted maze routing on layer l;
    Remove unnecessary jogs;
    Propagate the terminals of the uncompleted nets to layer $l + 1$;
    Rotate the routing area by 90 degrees;
    $l = l + 1$;
  end
end

Fig. 2. Overview of the SLICE router.

---

[2] Note that the scanning direction in the next layer will be orthogonal to the scanning direction in the current layer, which will also help to complete nets that require mainly vertical wires.

For each column-pair, the occupied grid points on the left column are called *start-points*. Clearly, each start-point is either a terminal propagated from the previous layer, or the endpoint of a partial routing solution computed in the previous channel. We denote a start-point n_i in the current layer by a triple $n_i = (x_i, y_i, net_i)$, where (x_i, y_i) is the coordinates of the point, and net_i is the net number of the point. For a start-point n_i, the terminal that it is to be connected to is called the *target* of n_i, denoted by $target(n_i)$. [3]

We shall concentrate our discussion on routing between a single column-pair. We begin with a list P_l that contains all the start-points on the left column and go through the following three steps to complete the planar routing. (1) For all start-points on the left column of the current column-pair, we generate a set S of weighted edges that connect these start-points to the right column. The weight for each edge represents the gain if we include this connection in the planar routing solution. (2) We compute the maximum weighted non-crossing matching S_{MWNCM} of S, which corresponds to the best topological planar routing solution between the current column-pair. We shall show that this step can be carried out optimally in $O(n \log n)$ time, where n is the number of edges in S. (3) Finally, we compute the physical routing solution based on the edges in S_{MWNCM}. The steps in the planar routing algorithm are illustrated in Fig. 3, where the net number for each terminal is given besides the terminal, and there are three column-pairs. Routing in the first column-pair has been completed, and we are processing the second column-pair. Fig. 3(a) shows the weighted edges extending from the start-points on the left column to the right column, Fig. 3(b) shows the edges selected in the maximum weighted non-crossing matching, and Fig. 3(c) shows the physical routing based on the selected edges. We now describe these steps in detail.

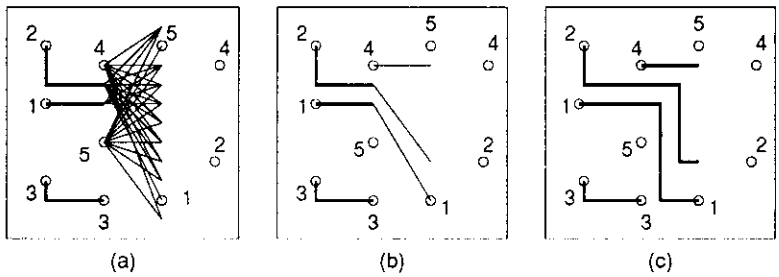

Fig. 3. Steps in planar routing for a column-pair. (a) Generate possible (topological) connections; (b) Compute a maximum weighted non-crossing matching; (c) Generate physical routing.

[3] As a preprocessing step of SLICE, we decompose each multi-terminal net into a set of two-terminal subnets based on the Prim's minimum spanning tree algorithm. Therefore, each start-point always has a well-defined target. The prior decomposition does not affect the routing quality very much since (1) most nets are two-terminal nets in MCM routing; (2) we allow the routes of two subnets of the same net to meet and form a Steiner point in our planar routing procedure.

3.2.1. Generating the weighted edges

Given a list of start-points P_l on the left column, we want to generate the set S of weighted edges that connects the start-points in P_l to the grid points on the right column. Conceptually, for a start-point $n_i = (x_i, y_i, net_i)$ in P_l, we can generate an edge from (x_i, y_i) to every grid point on the right column which is free or occupied by a terminal of net n_i. However, this may result in too many edges, and most of them will have very little chance of being selected in the maximum non-crossing matching in the next step. To conserve both memory and time, we use a simple heuristic, called *range reduction*, to reduce the number of edges that are generated. Clearly, the channel capacity C_{cap} bounds the density of the vertical segments that can be routed between a column-pair. Let y_{i+n} and y_{i-n} be the y-coordinates of the n-th start-point (*not* grid point) above and below n_i on the left column, respectively. Now if we assume that all the start-points on the left column will be routed, then the connection for each start-point above or below n_i will increase the channel density by one. Therefore, it is sufficient to generate edges whose right end-points are within the y-interval $[\, y_{i-C_{cap}}, y_{i+C_{cap}}]$. This is the reduced range where the edges for n_i can end on the right column.

The weight of each edge represents the gain if we include the edge in our planar routing solution. For each start-point n_i, let $n_j = target(n_i)$. We define the *preferred region* to be the y-interval on the right column defined by the y-coordinates of n_i and n_j. Clearly, if an edge from n_i ends within the preferred region, it does not increase the wirelength of net_i. So we assign a high weight, *weight_preferred*, to the edges ending in the preferred region. Moreover, if an edge from n_i ends exactly on n_j (when n_j is on the right column), we assign an even higher weight, *weight_completed*, to the edge in favor of completion of the net. For the edges that end outside the preferred region, we assign them a small weight *weight_outside*. In general, *weight_completed* > *weight_preferred* > *weight_outside*. We experimented with several weighting functions and the best choice is the following. We set *weight_completed* and *weight_preferred* to be two constants, and let *weight_outside* decrease linearly as the right end of the edge moves away from the preferred region.

3.2.2. Computing the maximum weighted non-crossing matching

The most important part of our planar routing algorithm is computing a topological planar routing solution between each column-pair. We begin with a set of weighted edges $S = \{s_1, s_2, \cdots, s_n\}$. Each edge in S represents a possible topological route that extends from a start-point to the right column. The weight for each edge represents the gain if we include this route in the planar routing solution. Each edge s_i is a four-tuple (l, r, w, net) where l is the y-coordinate of the left end of the edge, r is the y-coordinate of the right end of the edge, w is the weight of the edge, and *net* is the net number of the edge. Since we want to choose a set of best edges that can be routed on the current layer, we need to select a set of edges from S that are non-crossing and have the maximum total weight. This is the *maximum weighted non-crossing bipartite matching (MWNCM) problem*. However, in our formulation, we permit two edges in a non-crossing matching to share a common endpoint at the right column if they belong to the same net.

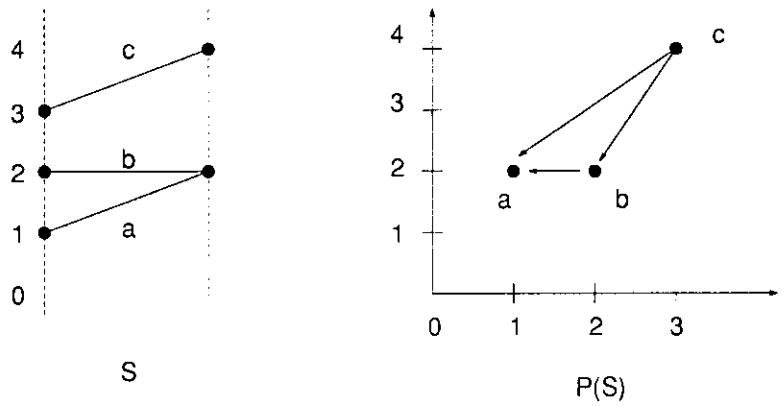

Fig. 4. Mapping the edges to points in the plane. Edges in $P(S)$ indicate the dominance relations (assuming the start-points 1 and 2 are of the same net).

In order to compute a MWNCM, we map each edge (l, r, w, net) to a point in the $x-y$ plane using the one-to-one mapping $(x,y) = (l,r)$, where (x,y) is the position of the point in the $x-y$ plane. Given two points $p_i = (x_i, y_i)$ and $p_j = (x_j, y_j)$ in the $x-y$ plane, p_i *dominates* p_j if (i) $x_i \geq x_j$ and $y_i > y_j$, or (ii) $x_i \geq x_j$ and $y_i = y_j$ and $net_i = net_j$ (note that net_i is the net that the corresponding edge of p_i belongs to). If condition (i) is satisfied, we say that p_i *strictly dominates* p_j; otherwise, when condition (ii) is satisfied, we say that p_i *laterally dominates* p_j. The dominance relations are illustrated in Fig. 4, where edge c strictly dominates edges a and b, and edge b laterally dominates edge a (assuming that a and b are of the same net). Clearly, if p_i dominates p_j, the two edges that are mapped to p_i and p_j are either strictly non-crossing or sharing the same endpoint on the right column when they are of the same net.[4] We define a *chain* among a set of points P in the $x-y$ plane, to be an ordered list of points $C = \{p_1, p_2, \cdots, p_m\}$ where each $p_k \in P$, and p_{k+1} dominates p_k for $k = 1, 2, \cdots, m-1$. We call p_m the *head-node* of the chain. We define the *weight* of a chain C, denoted by *weight* (C), to be the sum of the weights of the points in C. We define the *maximum-chain* $C_{\max}$ to be the chain that has the maximum weight among a given set of points. We then have the following results:

Lemma 1: The dominance relation is transitive.

Proof: Suppose that p_i dominates p_j and that p_j dominates p_k. There are four possibilities: (i) p_i strictly dominates p_j and p_j strictly dominates p_k; (ii) p_i strictly dominates p_j and p_j laterally dominates p_k; (iii) p_i laterally dominates p_j and p_j strictly dominates p_k; and (iv) p_i laterally dominates p_j and p_j laterally dominates p_k. It is straight forward to verify that in cases (i) - (iii), p_i strictly dominates p_k, and in case (iv) p_i laterally dominates p_k. Therefore, the dominance relation is transitive. □

[4] Note that when two edges of the same net share a right endpoint, a Steiner point is formed. So our method can generate Steiner routing trees automatically.

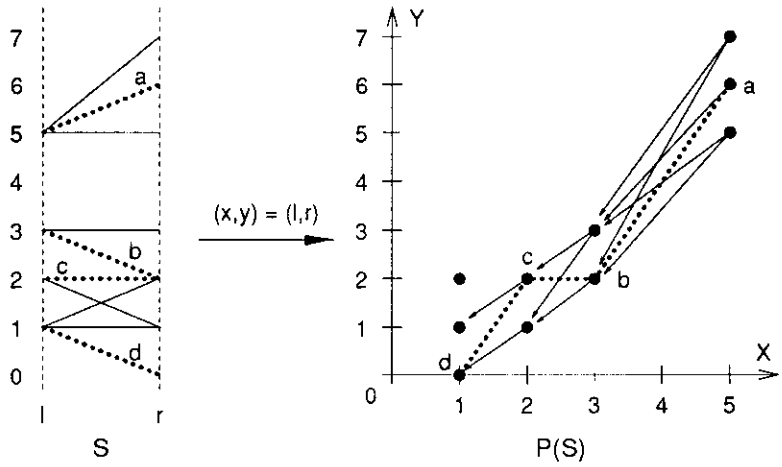

Fig. 5. A set of edges S and its corresponding point set $P(S)$. Note that start-points 2 and 3 are of the same net. The dominance relations are shown in $P(S)$ as thin arrows The MWNCM in S, and the corresponding maximum-chain in $P(S)$ are shown as dotted arrows.

Theorem 1: Let S be the set of edges between a column pair and $P(S)$ be the set of corresponding points on the $x-y$ plane. Then, $P(S)$ forms a partially ordered set. Moreover, the set of edges M in S is a maximum weighted non-crossing matching if and only if the corresponding points $P(M)$ form a maximum-chain in $P(S)$.

Proof It is easy to see that the dominance relation is *reflexive* (i.e., p_i dominates p_i itself) and *antisymmetric* (i.e., if p_i dominates p_j and p_j dominates p_i, then $p_i = p_j$). Also, according to Lemma 1, the dominance relation is transitive. Therefore, the point set $P(S)$ with the dominance relation forms a *partially ordered set* [14].

According to the definition of the dominance relation, it is easy to verify that two edges in M are non-crossing if and only if the two corresponding points in $P(M)$ are related by the dominance relation (i.e., one dominates the other). Therefore, the set of edges $M \subseteq S$ forms a non-crossing matching if and only if any two points in $P(M)$ are related by the dominance relation. Since $P(S)$ is a partially ordered set, any two points in $P(M)$ are related by the dominance relation is equivalent to that $P(M)$ is a chain in $P(S)$. Moreover, since the weight of the chain $P(M)$ is the same as the weight of the edge set M, we conclude that M is a maximum weighted non-crossing matching if and only if $P(M)$ is a maximum-chain in $P(S)$. □

A maximum-chain of a point set P can be computed as follows: We construct a directed graph G_P, called the *dominance graph*, in which each node represents a point in P, and add an edge (p_i,p_j) to G_P if and only if point p_i dominates point p_j. Fig. 5 shows a set of edges and the dominance graph on the corresponding point set (the edges implied by the transitive relation are omitted for clarity). It is not difficult to show that G_P thus constructed is a directed acyclic graph and a maximum-chain in P corresponds to a maximum weighted path in G_P. Since the maximum weighted path in a directed acyclic graph can be computed in $O(n^2)$ time, where n is the

number of nodes in the graph [15], we can compute a MWNCM in $O(n^2)$ time, where n is the number of edges we generated between a column-pair. The maximum weighted edges in S and the maximum weighted path in $P(S)$ are shown in Fig. 5 as dotted edges. However, since the procedure for computing a maximum weighted non-crossing matching will be used for every column-pair, we seek for a more efficient implementation. We have developed an $O(n\log n)$ time algorithm for computing the MWNCM based on a data structure called the priority search tree [16]. Before we describe the algorithm in detail, we first state a lemma:

Lemma 2 Suppose that each point in P has a positive weight. Then, if point p laterally dominates point q in a maximum-chain in P, there does not exist a point r such that p laterally dominates r and r laterally dominates q.

Proof: If such a point r exists, we can add it into the maximum-chain to get a chain of even larger weight, which leads to a contradiction. □

According to Lemma 2, if point p laterally dominates point q in a maximum-chain, then q is the first point of the same net left of p in the same row. This fact is used in the construction of a maximum-chain by our algorithm.

Let P be the corresponding point set of the given set of edges, we shall compute a maximum chain in P under the dominance relation. The points in P having the same x-coordinate form a *column*, and the points in P with the same y-coordinate form a *row*. Our algorithm processes the points on a row by row basis, and we process the points in the same row from left to right. This guarantees that when we are processing a point, all the points which are dominated by the current point have been processed already. During the execution of the algorithm, we maintain a binary priority search tree, called *PTREE*. Each leaf L of *PTREE* corresponds to a column occupied by a point in P, and it has three fields, $L.x$, $L.weight$, and $L.head$. The field $L.x$ stores the x-coordinate of the column. During the execution of our algorithm, assume that p is the highest point that we have processed so far at column $L.x$, then $L.weight$ is the weight of the maximum chain among the points that are *strictly* dominated by p or in the same column below p, and $L.head$ is the head node of that maximum chain (note that p may not be the head). We shall show how to maintain $L.weight$ and $L.head$ later on in the algorithm. Each internal node I of *PTREE* has a field $I.weight$ which records the largest weight of the leaves that are in the subtree *subtree*(I) rooted at I. (Clearly, if X and Y are the two children of I, then $I.weight = \max(X.weight, Y.weight)$.) Fig. 6 shows an instance of the *PTREE* and the point set in the $x-y$ plane. The weights for the leaf-nodes in *PTREE* and the points in P are also given in the Fig. 6. For example, when we start processing point q, we have $H.weight = 27$ and $H.head = a$ since the maximum chain among the points strictly dominated by q or in the same column below q is (a, b, c).

Our algorithm processes the points in P one row at a time from bottom to top, and processes the points in the same row from left to right, so we sort all the points according to their y-coordinates first and then x-coordinates. We maintain four fields for each point p: $p.weight$, $p.net$, $p.total\_weight$ and $p.next$. The fields $p.weight$ and $p.net$ store the weight and the net

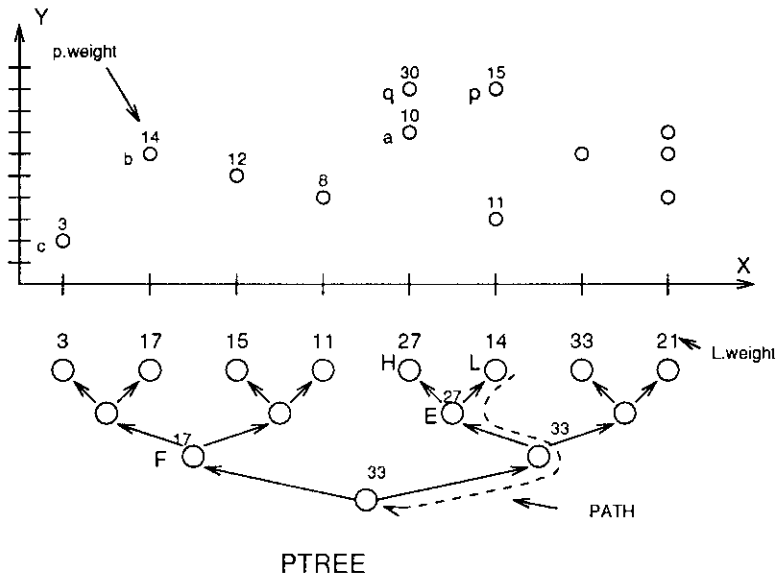

Fig. 6. Priority search tree (*PTREE*) used to compute the MWNCM.

number of the edge corresponding to the point, respectively. The field *p.total_weight* stores the weight of the maximum-chain C_p with p as its head node, and *p.next* points to the next point after p in C_p. Initially *p.total_weight* = 0 and *p.next* = *nil* for all points.

For each point p, *p.total_weight* can be determined as follows: let leaf L in *PTREE* correspond to the column where p is located. Let $PATH_L$ be the path from L to the root in *PTREE*. Let $l_1, l_2, \ldots, l_k$ be the roots of the left subtrees hanging from the path $PATH_L$ (i.e., l_i is the left child of some node in $PATH_L$). The algorithm searches for leaf L' such that

$$L'.weight = \max_{i=1}^{k} l_i.weight \tag{1}$$

Then, we have *p.total_weight* = *p.weight* + *L'.weight* and *p.next* = *L'.head*. This covers the case where p strictly dominates *p.next*. To cover the case that p laterally dominates *p.next*, we look for the point q in the same row as p with *q.net* = *p.net*. According to Lemma 2, we need only to consider such a q which is closest to p. If *p.total_weight* < *p.weight* + *q.total_weight*, then we set *p.total_weight* = *p.weight* + *q.total_weight* and *p.next* = q. Fig. 6 illustrates the computation of *p.total_weight* and *p.next* for the point p. The leaf node L in *PTREE* corresponds to the column where p is located. The path $PATH_L$ from L to the root is shown by the dashed line. The nodes F and H are the roots of the left subtrees hanging from $PATH_L$. According to Equation (1),

$$L'.weight = \max(F.weight,\ H.weight) = \max(17,27) = 27.$$

If point q is not in the same net as p, then

$$p.total\_weight = p.weight + L'.weight = 15 + 27 = 42.$$

If q and p are in the same net, then

$$p.total\_weight = \max(q.total\_weight,\ L'.weight) + p.weight = \max(47,\ 27) + 15 = 62.$$

(Note that $q.total\_weight = q.weight + F.weight = 30 + 17 = 47$, which has been computed when q was processed in the previous step.)

After we have processed a row of points, we update the entries in *PTREE*. (Note that we do not update the node leaf L immediately after we have processed the current point p, because we use $L.weight$ to record the weight of the maximum-chain among the points that are *strictly* dominated by p or are in the same column below p.) For each point q in the current row, let leaf L_q correspond to the column that q is located; if $q.total\_weight > L_q.weight$, then $L_q.weight = q.total\_weight$ and $L_q.head = q$. Furthermore, we update the weights of the internal nodes in $PATH_{L_q}$.

After all the points are processed, the point p_m with the largest *total_weight* is the head of the maximum chain in P and we can follow the $p_m.next$ field to get the rest of the maximum chain. Our algorithm is summarized in Fig. 7.

Theorem 2: Given the set S of n edges between a column-pair, the maximum weighted non-crossing matching can be computed in $O(n\log n)$ time.

Proof We shall show that our algorithm spends $O(\log n)$ time for processing each point p in P. Since *PTREE* is a binary priority search tree, it has depth $O(\log n)$. Thus, there are $O(\log n)$ left subtrees hanging from $PATH_L$. We can find out the subtree whose root, say l_i, has the maximum weight in $O(\log n)$. Moreover, we can find the leaf node L' in the subtree $subtree(l_i)$ with $L'.weight = l_i.weight$ in $O(\log n)$ time. (In order to find such a leaf node L', we start with l_i as the current node and always move the child who has the same weight as the current node. When we eventually hit a leaf node, we return it as L'.) Furthermore, to locate the point q, which is immediately laterally dominated by p, takes only constant time. (We preprocess each net and record for each point p the point which is immediately laterally dominated by p. This preprocessing can be done in linear time.) Therefore, $p.total\_weight$ can be computed in $O(\log n)$ time for each point p in P.

When p is the last point in a row, we update the leaf node L_q in *PTREE* for each point q in that row. Moreover, we update the weights of the internal nodes in $PATH_{L_q}$ (i.e. we start with L_q as the current node X. Let $parent(X)$ be the parent node of X in *PTREE*, then we update $parent(X).weight$ by $\max(parent(X).weight,\ X.weight)$. We assign $X = parent(X)$ and repeat the updating operation until $parent(X)$ becomes the root of *PTREE*). Since the length of each $PATH_{L_q}$ is $O(\log n)$, our algorithm spends $O(\log n)$ time updating *PTREE* for each point q in P.

Therefore, our algorithm spends at most $O(\log n)$ time processing each point p in P. Hence, the time complexity of our algorithm is bounded by $O(n\log n)$. □

Algorithm Compute MWNCM
  $P = \{p_1, p_2, \cdots, p_n\}$; /* The set of (x, y) points mapped from S */
  $PTREE$ = A priority search tree with leaves associated with the x-coordinates of the points in P;
  Sort P according to $p.y$ followed by $p.x$;

  foreach p **in** P
    let leaf L in $PTREE$ correspond to the column where p is located;
    let $PATH_L$ be the path from L to the root in $PTREE$;
    let $l_1, l_2, \ldots, l_k$ be the roots of the left subtrees hanging from $PATH_L$;
    locate the leaf node L' such that $L'.weight = \max_{i=1}^{k} l_i.weight$;
    let q be the rightmost point in the same row as p with $q.net = p.net$;
    if (q exists) and ($q.total\_weight > L'.weight$) **then**
      $p.total\_weight = q.total\_weight + p.weight$ and $p.next = q$;
    else
      $p.total\_weight = L'.\_weight + p.weight$ and $p.next = L'.head$;
    endif

    if (p is the last point in a row) **then**
      foreach q in the row **do**
        let leaf L_q correspond the column where q is located;
        if ($q.total\_weight > L_q.weight$) **then**
          $L_q.weight = q.total\_weight$;
          $L_q.head = q$;
          update the weights of the internal nodes in $PATH_{L_q}$;
        endif
      end /* of foreach */
    endif

  let p_m be the point with the largest $p.total\_weight$;
  construct the maximum chain $C_{\max}$ by following the $p_m.next$ field;
  S_{MWNCM} = the edges corresponding to the points in $C_{\max}$;
end

Fig. 7. Algorithm for computing the MWNCM.

We noticed a significant speed-up when the $O(n^2)$ time algorithm was replaced by the $O(n\log n)$ time algorithm for computing a maximum weighted non-crossing matching between each column-pair.

3.2.3. Physical routing

The solution we obtained from computing the maximum weighted non-crossing matching in the previous section gives us a set of edges, $S_{MWNCM} = \{s_1, s_2, \cdots, s_n\}$, where $s_i = (l_i, r_i, net_i)$, l_i and r_i being the y-coordinates of the left and right endpoints of the edge, and net_i is the net number of the edge. Each edge represents a connection between the two points,

(x_l, l_i) and (x_r, r_i) in the current routing plane, where x_l and x_r are the x-coordinates of the left and right columns, respectively. As the result of the physical routing, we end up with a list of endpoints of the routings, P_r, on the right column, which , together with the terminals that are on the right column, will form the new list of start-points on the new left column in the next iteration of column-pair routing. Because the edges in S_{MWNCM} are topological routing solutions, not all edges can be routed due to the channel capacity constraint. We add to P_{prop} the start-points whose edges failed to be routed.

We perform the routing separately on two classes of edges from S_{MWNCM} as defined below. Given an edge $s_i = (l_i, r_i, net_i)$, we say that s_i is a rising edge if $l_i < r_i$. Otherwise, we say that s_i is a falling edge (i.e. $l_i \geq r_i$). We group all the rising edges in S_{rise} and all the falling edges in S_{fall}. We also order the edges in S_{fall} in *increasing* y-coordinates, and order the edges in S_{rise} in *decreasing* y-coordinates. That is, if S_{rise} or $S_{fall} = \{s_1, s_2, \cdots, s_n\}$, then for S_{fall}, $l_i < l_{i+1}$ for $i = 0, \cdots, n-1$. Whereas for S_{rise}, $l_i \geq l_{i+1}$ for $i = 0, \cdots, n-1$. It is not difficult to show that the edges in S_{rise} and S_{fall} can be routed separately.

We perform the physical routing one edge at a time. We now describe the routing for S_{rise}. For each edge s_i in S_{rise}, we start routing from (x_l, l_i) in the routing plane, and route towards (x_r, r_i) in the following manner. We advance the routing along the y-axis upward until the routing is blocked, or if we have reached the y-coordinate r_i. Then we shall route one grid unit along the x-axis rightward if possible and repeat the routing along the y-axis. This process is repeated until one of the following three cases is encountered. (1) The connection is completed, (2) the routing has ended on the right column but did not reach (x_r, r_i), and (3) the routing has failed to reach the right column. For case (1), we simply add the start-point (x_r, r_i, net_i) to P_r. For case (2), we add the start-point with the new y-coordinate, (x_r, new_r_i, net_i) to P_r, where r_new_i is the y-coordinate of the end-point of the physical routing on the right column. For case (3) we remove any partial routing that we might have added between the column-pair, and add the start-points $n_i = (x_l, l_i, net_i)$, and $target(n_i)$ to the list of terminals P_{prop} to be propagated to the next layer. Fig. 8 illustrates these cases. The left side of Fig. 8 shows four rising edges in the MWNCM named *a, b, c, d*. Terminals *t* 1 and *t* 2 are of other nets. Routing for edges *a* and *b* are completed. Edge *c* is routed to the right column but at a different y-coordinate. That is, we have altered the topological solution since the end-point of the routing does not correspond to the end-point of the edge in MWNCM. However, we fail to route edge *d* because of the blocking terminal on the right column. In this case, both the start-point of the edge *d* and its target will be added to P_{prop}. For edges in S_{fall}, the procedure is similar except that routing along the y-axis is always downward.

3.3. Pin redistribution

Another feature of the SLICE router is that it redistributes terminals during the routing process to avoid local congestion. Unlike the pin redistribution algorithm presented in [12], our pin redistribution process is interleaved with the routing process. At the end of planar routing of each layer, a pin redistribution step is performed. Since the planar routing for a net is blocked only when it encounters some obstacles or other routings, the terminals in P_{prop} tend to be clustered.

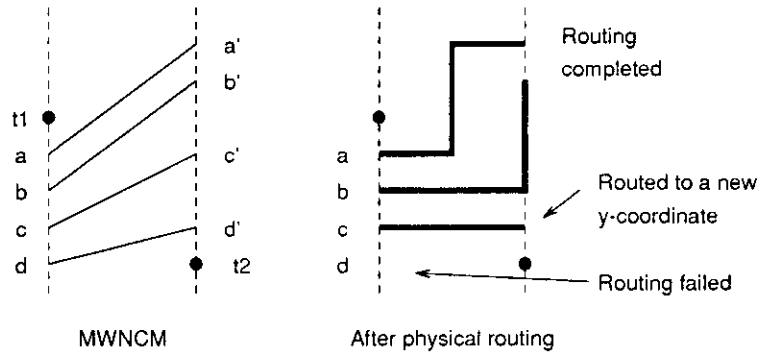

Fig. 8. Example of physical routing.

This will make the routing on subsequent layers difficult because the routings will be congested around the clustered terminals. To reduce the routing congestion, we want to ensure that the terminals in P_{prop} are are evenly distributed. We define the *terminal density* of a given column to be the number of terminals in that column. Then to reduce the routing congestion, we try to move the terminals in P_{prop} such that terminal densities are roughly equal among all the columns[5]. Furthermore, we should try to move the terminals in such a way so that the increase in wirelength is minimized. Our pin redistribution algorithm processes the terminals one at a time, moving the terminals horizontally to a column with the lowest terminal density. For a given terminal of net n, the possible columns that it can be moved to is restricted to be in the range $[x(n)_l - slack, x(n)_r + slack]$, where $x(n)_l$ and $x(n)_r$ are the x-coordinates of the leftmost and rightmost terminals of net n, and *slack* is a small constant. Experimental results show that the pin redistribution algorithm consistently improves the utilization of the routing resources.

3.4. Restricted maze routing

Our planar algorithm will produce routing segments extending predominantly in the scanning direction. Therefore, many start-points may not be routed because they are lined up almost vertically. To complement the planar routing, we use a restricted maze router to complete as many left over nets as possible.

To conserve memory, we restrict the maze-routing to within two routing layers. Moreover, we restrict the range of the maze router to a thin "vertical slice" of the routing region since we are primarily interested in vertical connections. Typically, the maze range is 10% of the width of the routing region.

3.5. Jog removal

Since the planar routing algorithm does not penalize the formation of wiring jogs, the completed routings may contain many unnecessary jogs. Therefore, a clean up phase is necessary to remove these jogs to improve the quality of the planar routing solution.

[5] Since the routing direction for the next layer is vertical, the terminals that are on the same column often block each other. Therefore, reduction in terminal density at each column leads to better routing results.

We identify two kinds of jogs. We call *simple jogs* to be those that can be eliminated by a moving a single wire segment as shown in Fig. 9. Otherwise, the remaining jogs are called the jogs *complex jogs*, where more then one wire segments need to be moved to eliminate a jog as shown in Fig. 10. SLICE tries to remove the simple jogs first, then it tries to remove the remaining complex jogs. Both algorithms are based on the efficient plane sweeping technique used extensively in computational geometry [17]. Experimental results show that on the average, more than 49% of the jogs can be removed by our algorithms.

4. Experimental Results

We implemented SLICE on the Sun workstations using the C language. The following experimental results were recorded on a Sun SPARC station II with 32MB of memory. We tested the program on five examples shown in Table 1. The examples test1, test2, and test3, are generated with random netlists. Examples mcc1 and mcc2 were industrial MCM routing examples provided by MCC. Example mcc2 is a supercomputer with 37 VHSIC gate arrays.

Example	number of chips	number of nets	number of pins	size of substrate (mm^2)	pitch (um)	grid size
test1	4	500	1000	22.5 x 22.5	75	300 x 300
test2	9	957	1914	30 x 30	75	400 x 400
test3	9	1254	2508	37.5 x 37.5	75	500 x 500
mcc1	6	802	2495	45 x 45	75	599 x 599
mcc2	37	7118	14570	152.4 x 152.4	75	2032 x 2032

Table 1: Characteristics of examples

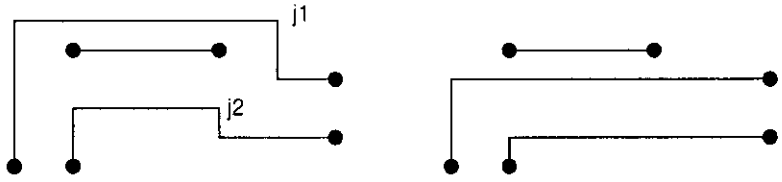

Fig. 9. Removal of simple jogs by moving horizontal segments downward.

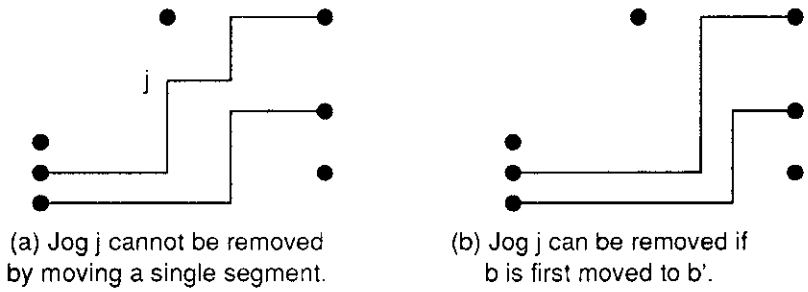

Fig. 10. Example of a complex jog.

The routing results obtained by SLICE on these examples are given in Table 2. The lower bound on wire length for each net is computed by the half perimeter of the bounding rectangle that encloses all the terminals in the net. This is a conservative lower bound for multi-terminal nets. It can be seen that SLICE uses at most 9% more than this lower bound for all examples except mcc1 [6].

Example	number of layers	number of vias	number of jogs	wire length			run time
				lower bound	SLICE	ratio	(hr:min)
test1	5	2013	3453	102238	109092	1.067	0:02
test2	6	5271	9656	265000	286723	1.082	0:06
test3	6	6892	13552	426308	459046	1.077	0:12
mcc1	5	6386	11215	339226	402258	1.186	0:12
mcc2	7	47864	108321	5622935	5902818	1.050	8:15

Table 2: Characteristics of solutions

Table 3 shows that a large percentage of the nets are completed within the first few routing layers. For all cases, more than 80% of the nets are completed within the first 4 routing layers.

Example	% of nets completed in x layers						
	1	2	3	4	5	6	7
test1	4.8	52.8	80.2	97.6	100.0		
test2	2.5	27.6	55.8	82.7	96.3	100.0	
test3	2.3	30.1	56.9	84.9	97.3	100.0	
mcc1	12.2	53.1	82.8	98.0	100.0		
mcc2	1.0	33.1	59.5	81.2	93.5	99.0	100

Table 3: Distribution of completed nets

Table 4 shows the effect of the two jog removal algorithms. Note that each of the individual jog removal algorithms may remove both kinds of jogs, thus the total jogs removed by applying both algorithms are less than the sum of the jogs removed by applying the two individual algorithms independently.

[6] There are many multi-terminal nets in mcc1. Since the lower bound given by the half perimeter of the bounding box is not tight for multi-terminal nets of size 4, our lower bound for mcc1 is considerably smaller than the optimal wirelength. In fact, it is commonly believed that the wirelength of a minimum Steiner tree for a multi-terminal net is at most 88% of the wirelength of a minimum spanning tree on average [18], which leads to a new wirelength lower bound of 362497 for mcc1. The result by the SLICE router is only 11.0% more than the new lower bound.

Example	Total number of jogs			
	Without jog removal	Algo. to remove Simple Jogs	Algo. to remove Complex Jogs	Both Algorithms
test1	6732	3785	4015	3453
test2	18519	10991	11328	9656
test3	25725	15944	16079	13552
mcc1	20399	12260	12999	11215
mcc2	249551	131681	126173	108321

Table 4: Effects of jog removal algorithms

Table 5 shows the effect of the pin redistribution algorithm. The algorithm consistently reduces the number of layers needed to complete the routing at the expense of a slight increase (2.5%) in the total wirelength. The impact of the pin redistribution algorithm on the number of vias and jogs is usually small. On average, the number of jogs is increased by 2.5% and the number of vias is increased by 0.6%.

Example	no. of layers		no. of vias		no. of jogs		total wirelength	
	with	without	with	without	with	without	with	without
test1	5	6	2013	2025	3453	3366	109092	107247
test2	6	7	5271	5268	9656	9230	286723	279542
test3	6	7	6892	6821	13552	13326	459046	445208
mcc1	5	6	6386	6120	11215	11183	402258	388078
mcc2	7	9	47864	48769	108321	104606	5902818	5807699

Table 5: Effects of pin redistribution algorithm.

We also compare our results with a general 3D maze router in Table 6. The 3D maze router uses a reserved layer model [7] [8], in which the horizontal wires and vertical wires are routed in different layers. The 3D maze router was not able to run on mcc2 on our system due to the large size of the example. On the average, SLICE is more than six times faster than the 3D maze router, and uses 29% fewer vias than the 3D maze router. However, the number of layers used by SLICE is generally more than the 3D maze router. But as shown in Table 3, the last few layers in the SLICE solutions are very sparse.

[7] The 3D maze router using the reserved layer model performed much better than the one using the un-reserved layer model. For example, in our experiment, the number of layers required for example mcc1 is 17 with the un-reserved layer model versus 5 layer with the reserved layer model.

[8] We tried both the reserved layer and the un-reserved layer model two-layer maze router in SLICE but the difference in the results is insignificant. For all the test cases, the results by SLICE reported in Table 2 to 6 are based on the un-reserved layer model two-layer maze router.

Example	number of layers		number of vias		number of jogs		total wire length		run time (hr:min)	
	SLICE	maze	SLICE	maze	SLICE	maze	SLICE	maze	SLICE	maze
test1	5	4	2013	2975	3453	421	109092	107908	0:02	0:08
test2	6	4	5271	7127	9656	892	286723	273642	0:06	0:48
test3	6	4	6892	9347	13552	1094	459046	441552	0:12	1:40
mcc1	5	5	6386	8794	11215	1244	402258	397221	0:12	0:59
mcc2	7	-	47864	-	108321	-	5902818	-	8:15	-

Table 6: Comparison with maze router

Another advantage that SLICE has over the 3D maze router is its low memory requirement. For the example mcc2 (a supercomputer with 37 gate arrays), in order to store the entire grid of size 7×2032×2032, the 3D maze router needs 110MB of memory (assuming that we use four bytes for each grid point to store the net number, routing cost, etc.) That is why the 3D maze router failed to route the example on our system. However, using a maze routing range of 10%, at any time, the working space of SLICE is only 2×10%×2032×2032 = 3.3MB of memory. So SLICE successfully produced a satisfactory solution. Furthermore, if the routing pitch for the same example is reduced by a factor of two, the 3D maze router will require 441MB of memory whereas SLICE will require only 13.2MB of working memory. Clearly, for the next generation of dense MCM design, the 3D maze router will face more severe memory limitation, and the advantage of SLICE will become much more significant.

5. Conclusions and Future Extensions

In this paper, we presented a fast multilayer general area router named SLICE for MCM Designs. The routing result of the SLICE router is independent of net ordering and uses fewer vias. The total wirelength produced by SLICE is only a few percents away from the optimal. Compared with a general 3D maze router, with a small increase in the number of routing layers, SLICE runs more than six times faster, uses 29% fewer vias, and requires far less memory.

The SLICE router can also handle 45 degree routing. After we obtain a maximum weighted non-crossing matching, we can use a more sophisticated procedure to map the topological routing solution into a physical routing solution which allows 45 degree routing. The SLICE router can handle arbitrary obstacle in the routing region as well since it can avoid generating edges whose end-points are on the obstacles.

Although the SLICE router reduces the total number of vias significantly, it might be possible the some individual nets have high via counts. We are in the process of developing a MCM router which can bound the number of vias used for every net in order to achieve predictable performance. We also hope to take some performance issues (such as coupling and reflection) into consideration in our design of the next generation of MCM router.

6. Acknowledgments

The authors thank Prof. C. K. Cheng at UCSD and Deborah Cobb at MCC for providing the two MCM industrial examples. This research is partially supported by the National Science Foundation under grant MIP-9110511.

7. References

[1] Preas, B., M. Pedram, and D. Curry, "Automatic Layout of Silicon-On-Silicon Hybrid Packages," *ACM/IEEE 26th Design Automation Conference*, pp. 394-399, 1989.

[2] Bakoglu, H. B., *Circuits, Interconnections, and Packaging for VLSI*, Addison-Wesley Publishing Company, Menlo Park, California (1990).

[3] Herrell, D., "Multichip Module Technology At MCC," *IEEE Int'l Symposium on Circuits and Systems*, pp. 2099-2103, 1990.

[4] Blodgett, A. J. and D. R. Barbour, "Thermal conduction module: a high performance multilayer ceramic package," *IBM Journal of Research and Development*, Vol. **26**, pp. 30-36, Jan. 1982.

[5] Blodgett, A. J., "Microelectronic packaging," *Scientific American*, pp. 86-96, July 1983.

[6] Hanafusa, A., 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.

[7] Miracky, R. et al, "Technology for Rapid Prototyping of Multi-Chip Modules," *IEEE Int'l Conf. on Computer Design*, pp. 588-591, 1991 .

[8] Ho, J. M., 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.

[9] Dai, W. M., R. Kong, J. Jue, and M. Sato, "Rubber Band Routing and Dynamic Data Representation," *IEEE Int'l Conf. on Computer-Aided Design*, pp. 52-55, 1990.

[10] Dai, W. M., T. Dayan, and D. Staepelaere, "Topological Routing in SURF: Generating a Rubber-Band Sketch," *ACM/IEEE 28th Design Automation Conference*, pp. 41-44, 1991.

[11] Dai, W. M., R. Kong, and M. Sato, "Routability of a Rubber-Band Sketch," *ACM/IEEE 28th Design Automation Conference*, pp. 45-48, 1991.

[12] Cho, J. D. and M. Sarrafzadeh, "The Pin Redistribution Problem in Multi-Chip Modules," *Proc. IEEE ASIC'91*, pp. P9-2.1, 1991.

[13] Shambrook, K., "Overview of Multichip Module Technologies," *Multichip Module Workshop*, pp. 1-9, 1991.

[14] Liu, C. L., *Elements of Discrete Mathematics*, McGraw-Hill Book Co., New York (1977).

[15] Reingold, E., J. Nievergelt, and N. Deo, *Combinatorial Algorithms: Theory and Practice*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey (1977).

[16] Tarjan, R. E., *Data Structures and Network Algorithms*, Society for Industrial and Applied Mathematics, Philadelphia, Pennsylvania (1983).

[17] Preparata, F. P. and M. I. Shamos, *Computational Geometry*, Springer-Verlag, New York (1985).

[18] Bern, M. W. and M. d. Carvalho, "A Greedy Heuristic for the Rectilinear Steiner Tree Problem," in *UC Berkeley Computer Science Department Tech. Report 87-306*, , Berkeley, CA (1987).

rabbit:cong

paper.cas.ps

Tue Aug 18 21:07:21 1992

37 / 37 NT CA7501864 Hilo

37 rabbit:cong Job: paper.cas.ps Date: Tue Aug 18 21:07:21 1992

37 rabbit:cong Job: paper.cas.ps Date: Tue Aug 18 21:07:21 1992

37 rabbit:cong Job: paper.cas.ps Date: Tue Aug 18 21:07:21 1992

37 rabbit:cong Job: paper.cas.ps Date: Tue Aug 18 21:07:21 1992

TR 9200011

same abstract