A Fast Multilayer General Area Router for MCM Designs
Kei-Yong Khoo and Jason Cong

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 (3-D) 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 3-D 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 noncrossing 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 3-D 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 3-D maze router works on the entire 3-D routing grid. Therefore, SLICE can successfully produce solutions for large MCM routing examples where 3-D maze routers fail due to insufficient memory.

I. INTRODUCTION

As VLSI fabrication technology advances, interconnection and packaging (P/I) technologies have become a bottleneck in system performance [1]–[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 SCP’s 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, MCM’s may have far more interconnection layers than IC’s. For example, the MCM developed for the IBM 3081 mainframe has 33 layers of molybdenum conductors (including one bonding layer, five distribution layers, 16 interconnection layers, eight voltage reference layers, and three power distribution layers [4], [5]. Fujitsu’s latest supercomputer, the VP-2000, uses a ceramic substrate with more than 50 interconnection layers [6]. Moreover, unlike routing in IC’s, 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 (3-D) 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 3-D 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, 3-D 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 $174 \times 174 \text{mm}^2$; this results in a routing grid of $2032 \times 2032$ for a single layer! It is certainly not a trivial task to store such a grid for each layer and search in it efficiently.

1Beside 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.
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 predetermine 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]–[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, 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 3-D 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 noncrossing 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 3-D maze router. On average, both routers use about the same total wirelength, but the 3-D maze router is six 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 3-D maze router works on the entire 3-D routing grid. Therefore, SLICE can successfully produce solutions for large MCM routing examples where 3-D maze routers fail due to the memory requirement.

The remainder of this paper is organized as follows. Section II formulates the multilayer MCM routing problem. In Section III, 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 IV. Finally, we discuss the extension of our work in Section V.

II. 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 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° 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 nonadjacent 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 MCM 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 degrade the signal’s fidelity by introducing impedance discontinuity in signal paths and 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.

III. Description of the Algorithm

In this section, we present our fast multilayer 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.
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 \( C_{\text{cap}} = x_r - x_l \). During planar routing in the current layer, the terminals of the uncompleted nets are put in the list \( P_{\text{prop}} \). These terminals will be propagated to the next routing layer.

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 \( n_i = (x_i, y_i, \text{net}_i) \), where \((x_i, y_i)\) is the coordinates of the point, and \( \text{net}_i \) 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 \( \text{target}(n_i) \).

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 noncrossing matching \( S_{\text{MWNCM}} \) 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 \( S_{\text{MWNCM}} \). 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 max-

---

\(^2\)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 main-
3.2.2. Computing the Maximum Weighted Noncrossing 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 \( S = \{s_1, s_2, \ldots, s_n\} \). 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 noncrossing bipartite matching (MWNBM) problem. However, in our formulation, we permit two edges in a noncrossing matching to share a common endpoint at the right column if they belong to the same net.

In order to compute a MWNBM, 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 \( y_i \geq y_j \), or ii) \( x_i = x_j \) and \( y_i \geq y_j \) and \( net_i = net_j \), (note that \( net \) 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 noncrossing or sharing the same endpoint on the right column when they are of the same net. We define a chain among a set of points \( P \) in the \(x-y\) plane, to be an ordered list of points \( C = \{p_1, p_2, \ldots, p_m\} \) where each \( p_k \in P \), and \( p_{k+1} \) dominates \( p_k \) for \( k = 1, 2, \ldots, m - 1 \). 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 \( C_{\text{max}} \) 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 straightforward 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.

**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 noncrossing 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 noncrossing 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 noncrossing 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 and are 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 noncrossing 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 corresponding graph on the corresponding point set (the edges implied by the transitive relation are omitted for clarity). It is not difficult to show that $G_P$ 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 noncrossing 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 node). 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 $\text{subtree}(I)$ rooted at $I$. (Clearly, if $X$ and $Y$ are the two children of $I$, then $I.\text{weight} = \max(X.\text{weight}, Y.\text{weight}).$) 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 $H.\text{weight} = 27$ and $H.\text{head} = a$ 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 $p$: $p.\text{weight}$, $p.\text{net}$, $p.\text{total}._\text{weight}$, and $p.\text{next}$. The fields $p.\text{weight}$ and $p.\text{net}$ store the weight and the net number of the edge corresponding to the point, respectively. The field $p.\text{total}._\text{weight}$ stores the weight of the maximum-chain $C_p$ with $p$ as its head node, and $p.\text{next}$ points to the next point after $p$ in $C_p$. Initially $p.\text{total}._\text{weight} = 0$ and $p.\text{next} = \text{nil}$ for all points.

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

\[
L'.\text{weight} = \max_{i=1}^{k} l_i.\text{weight}. \tag{1}
\]

Then, we have $p.\text{total}.\text{weight} = p.\text{weight} + L'.\text{weight}$ and $p.\text{next} = L'.\text{head}$. This covers the case where $p$ strictly dominates $p.\text{next}$. To cover the case that $p$ laterally dominates $p.\text{next}$, we look for the point $q$ in the same row as $p$ with $q.\text{net} = p.\text{net}$. According to Lemma 2, we need only to consider such a $q$ which is closest to $p$. If $p.\text{total}.\text{weight} < p.\text{weight} + q.\text{total}.\text{weight}$, then we set $p.\text{total}.\text{weight} = p.\text{weight} + q.\text{total}.\text{weight}$ and $p.\text{next} = q$. Fig. 6 illustrates the computation of $p.\text{total}.\text{weight}$ and $p.\text{next}$ for the point $p$. The leaf node $L$ in PTREE corresponds to the column where $p$ is located. The path $\text{PATH}_L$ 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 $\text{PATH}_L$. According to (1),

\[
L'.\text{weight} = \max (F.\text{weight}, H.\text{weight})
= \max (17, 27) = 27.
\]

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

\[
p.\text{total}.\text{weight} = p.\text{weight} + L'.\text{weight} = 15 + 27 = 42.
\]

If $q$ and $p$ are in the same net, then

\[
p.\text{total}.\text{weight} = \max (q.\text{total}.\text{weight}, L'.\text{weight})
+ p.\text{weight} = \max (47, 27) + 15 = 62.
\]

(Note that $q.\text{total}.\text{weight} = q.\text{weight} + F.\text{weight} = 30 + 17 = 47$, which was 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 $L.\text{weight}$ 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 $q.\text{total}.\text{weight} > L_q.\text{weight}$, then $L_q.\text{weight} = q.\text{total}.\text{weight}$ and $L_q.\text{head} = q$. Furthermore, we update the weights of the internal nodes in $\text{PATH}_L_q$.

After all the points are processed, the point $p_{\text{max}}$ with the largest total.weight is the head of the maximum chain in $P$ and we can follow the $p_{\text{max}}.\text{next}$ 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 noncrossing 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 $\text{PATH}_L$. We can find out the subtree whose root, say $l_k$, has the maximum weight in $O(\log n)$. Moreover, we can find the leaf node $L'$ in the subtree $\text{subtree}(l_k)$ with $L'.\text{weight} = l_k.\text{weight}$ in $O(\log n)$ time. (In order to find such a leaf node $L'$, we start with $l_k$ as the current node and always move to 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, $p.\text{total}.\text{weight}$ 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. More-
Algorithm Compute MWNCM

\[ P = (p_1, p_2, \ldots, p_{|V|}) \]  
\[ P.TREE = \text{a priority search tree with leaves associated with the x-coordinates of the points in } P. \]

Sort \( P \) according to \( x \) y followed by \( x \).

For each \( p \in P \).

Let \( l_n \) be the leaf of \( P.TREE \) corresponding to the column \( n \).  
Let \( P.TREE \) be the path from \( l_n \) to the root in \( P.TREE \).

Let \( l_i, l_{i+1}, \ldots, l_{i+q} \) be the roots of the left sub-trees hanging from \( P.TREE \).

Locate the node \( L \) such that \( L.weight = \max \{ w_1, \ldots, w_q \} \).

Let \( q \) be the minimum \( q \) such that \( p \) is in the same row as \( L \).

If \( q < q_{\text{max}} \), \( p \) is a new node.

Let \( p \) be the internal node \( \{ q_{\text{max}}, \ldots, q \} \).  
If \( q_{\text{max}}.weight > L.weight \).

Let \( p \) be the internal node \( \{ q_{\text{max}}, \ldots, q \} \).

End for each \( p \).

Fig. 7. Algorithm for computing the MWNCM.

channel capacity constraint. We add to \( P_{\text{prop}} \) the start-points whose edges failed to be routed.

We perform the routing separately on two classes of edges from \( S_{\text{MWNCM}} \) as defined below. Given an edge \( s_i = (l_i, r_i, \text{net}_i) \), 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 \( S_{\text{rise}} \) and all the falling edges in \( S_{\text{fall}} \). We also order the edges in \( S_{\text{rise}} \) in increasing y-coordinates, and order the edges in \( S_{\text{rise}} \) in decreasing y-coordinates. That is, if \( S_{\text{rise}} \) or \( S_{\text{fall}} \) is \( \{ s_1, s_2, \ldots, s_q \} \), then for \( s_{i_{\text{left}}} \), \( l_i \leq l_{i+1} \) for \( i = 0, \ldots, n - 1 \), whereas for \( S_{\text{rise}} \), \( l_i \geq l_{i+1} \) for \( i = 0, \ldots, n - 1 \). It is not difficult to show that the edges in \( S_{\text{rise}} \) and \( S_{\text{fall}} \) can be routed separately.

We perform the physical routing one edge at a time. We now describe the routing for \( S_{\text{rise}} \). For each edge \( s_i \) in \( S_{\text{rise}} \), we start routing from \( (x_i, l_i) \) in the routing plane, and route towards \((x_i, r_i)\) 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 \((x_i, r_i)\), and 3) the routing has failed to reach the right column. For the case 1), we simply add the start-point \((x_i, r_i, \text{net}_i)\) to \( P \). For case 2), we add the start-point with the new y-coordinate, \((x_i, \text{new}_{r_i}, \text{net}_i)\) to \( P \), where \( \text{new}_{r_i} \) 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 \( n_i = (x_i, l_i, \text{net}_i) \), and \( \text{target}(n_i) \) to the list of terminals \( P_{\text{prop}} \) 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 \( r_1 \) and \( r_2 \) 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 \( P_{\text{prop}} \). For edges in \( S_{\text{fall}} \) 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 routing, the terminals in \( P_{\text{prop}} \) tend to be clustered. 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 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 the 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 \([x(n) - slack, x(n) + slack]\), where \(x(n)\) and \(x(n)\) 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.

We identify two kinds of jogs. We call simple jogs to be those that can be eliminated by moving a single wire segment as shown in Fig. 9. Otherwise, the remaining jogs are called the complex jogs, where more than 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 47% of the jogs can be removed by our algorithms.

### IV. 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 32 MB of memory. We tested the program on five examples shown in Table I. 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.

The routing results obtained by SLICE on these examples are given in Table II. 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 multiterminal
TABLE II

CHARACTERISTICS OF SOLUTIONS

<table>
<thead>
<tr>
<th>Example</th>
<th>number of layers</th>
<th>number of vias</th>
<th>number of jogs</th>
<th>wire length</th>
<th>run time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>lower bound</td>
<td>SLICE</td>
</tr>
<tr>
<td>test1</td>
<td>5</td>
<td>2013</td>
<td></td>
<td>3453</td>
<td>102238</td>
</tr>
<tr>
<td>test2</td>
<td>6</td>
<td>5247</td>
<td></td>
<td>9666</td>
<td>265000</td>
</tr>
<tr>
<td>test3</td>
<td>6</td>
<td>6592</td>
<td></td>
<td>13552</td>
<td>265038</td>
</tr>
<tr>
<td>mcc1</td>
<td>5</td>
<td>6386</td>
<td></td>
<td>11215</td>
<td>339267</td>
</tr>
<tr>
<td>mcc2</td>
<td>7</td>
<td>47751</td>
<td></td>
<td>107888</td>
<td>5362181</td>
</tr>
</tbody>
</table>

TABLE III

DISTRIBUTION OF COMPLETED NETS

<table>
<thead>
<tr>
<th>Example</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>test1</td>
<td>4.8</td>
<td>52.8</td>
<td>80.2</td>
<td>97.6</td>
<td>100.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>test2</td>
<td>2.5</td>
<td>27.6</td>
<td>55.8</td>
<td>82.7</td>
<td>96.3</td>
<td>100.0</td>
<td></td>
</tr>
<tr>
<td>test3</td>
<td>2.3</td>
<td>30.1</td>
<td>56.9</td>
<td>84.9</td>
<td>97.3</td>
<td>100.0</td>
<td></td>
</tr>
<tr>
<td>mcc1</td>
<td>12.2</td>
<td>53.1</td>
<td>82.8</td>
<td>98.0</td>
<td>100.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mcc2</td>
<td>1.6</td>
<td>33.9</td>
<td>60.9</td>
<td>81.2</td>
<td>94.2</td>
<td>99.3</td>
<td>100</td>
</tr>
</tbody>
</table>

TABLE IV

EFFECTS OF JOG REMOVAL ALGORITHMS

<table>
<thead>
<tr>
<th>Example</th>
<th>Without jog removal</th>
<th>Algo. to remove Simple Jogs</th>
<th>Algo. to remove Complex Jogs</th>
<th>Both Algorithms</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Total number of jogs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>test1</td>
<td>6732</td>
<td>3785</td>
<td>4015</td>
<td>3453</td>
</tr>
<tr>
<td>test2</td>
<td>18519</td>
<td>10991</td>
<td>11328</td>
<td>9656</td>
</tr>
<tr>
<td>test3</td>
<td>25725</td>
<td>15944</td>
<td>16079</td>
<td>13552</td>
</tr>
<tr>
<td>mcc1</td>
<td>20399</td>
<td>12260</td>
<td>12999</td>
<td>11215</td>
</tr>
<tr>
<td>mcc2</td>
<td>241844</td>
<td>129127</td>
<td>12256                  107888</td>
<td></td>
</tr>
</tbody>
</table>

nets. It can be seen that SLICE uses at most 9% more than this lower bound for all examples except mcc1.4

Table III 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.

Table IV 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.

Table V shows the effect of the pin redistribution algorithm. The algorithm consistently reduces the number of jogs needed to complete the routing at a slight increase (2.6%) 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 0.3%.

We also compare our results with a general 3-D maze router in Table VI. The 3-D maze router uses a reserved layer model,7,8 in which the horizontal wires and vertical wires are routed in different layers. The 3-D 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 3-D maze router, and uses 29% fewer vias than the 3-D maze router. However, the number of layers used by SLICE is generally more than the 3-D maze router. But as shown in Table III, the last few layers in the SLICE solutions are very sparse.

Another advantage that SLICE has over the 3-D 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 3-D maze router needs 110 MB of memory (assuming that we use four bytes for each grid point to store the net number, routing cost, etc.) That is why the 3-D 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 = 3.3 MB 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 3-D maze router will require 441 MB of memory whereas SLICE will require only 13.2 MB of working memory. Clearly, for the next generation of dense MCM design, the 3-D maze router will face more severe memory limitation, and the advantage of SLICE will become much more significant.

V. 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 3-D 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.
TABLE V

<table>
<thead>
<tr>
<th>Example</th>
<th>no. of layers</th>
<th>no. of vias</th>
<th>no. of jogs</th>
<th>total wirelength</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>with</td>
<td>without</td>
<td>with</td>
<td>without</td>
</tr>
<tr>
<td>test1</td>
<td>5</td>
<td>6</td>
<td>2013</td>
<td>2025</td>
</tr>
<tr>
<td>test2</td>
<td>6</td>
<td>7</td>
<td>5271</td>
<td>5268</td>
</tr>
<tr>
<td>test3</td>
<td>6</td>
<td>7</td>
<td>6892</td>
<td>6821</td>
</tr>
<tr>
<td>mocc1</td>
<td>5</td>
<td>6</td>
<td>6386</td>
<td>6120</td>
</tr>
<tr>
<td>mocc2</td>
<td>7</td>
<td>9</td>
<td>47751</td>
<td>49475</td>
</tr>
</tbody>
</table>

TABLE VI

<table>
<thead>
<tr>
<th>Example</th>
<th>number of layers</th>
<th>number of vias</th>
<th>number of jogs</th>
<th>total wire length</th>
<th>run time (hr:min)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SLICE</td>
<td>maze</td>
<td>SLICE</td>
<td>maze</td>
<td>SLICE</td>
</tr>
<tr>
<td>test1</td>
<td>5</td>
<td>4</td>
<td>2013</td>
<td>2975</td>
<td>3453</td>
</tr>
<tr>
<td>test2</td>
<td>6</td>
<td>4</td>
<td>5271</td>
<td>7127</td>
<td>9656</td>
</tr>
<tr>
<td>test3</td>
<td>6</td>
<td>4</td>
<td>6892</td>
<td>9347</td>
<td>13552</td>
</tr>
<tr>
<td>mocc1</td>
<td>5</td>
<td>5</td>
<td>6386</td>
<td>8794</td>
<td>12115</td>
</tr>
<tr>
<td>mocc2</td>
<td>7</td>
<td>—</td>
<td>47751</td>
<td>—</td>
<td>107888</td>
</tr>
</tbody>
</table>

The SLICE router can also handle 45° routing. After we obtain a maximum weighted noncrossing matching, we can use a more sophisticated procedure to map the topological routing solution into a physical routing solution which allows 45° 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 that some individual nets have high vias 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.

ACKNOWLEDGMENTS

The authors thank Prof. C. K. Cheng at UCSD and Deborah Cobb at MCC for providing the two MCM industrial examples.

REFERENCES

Jason Cong received the B.S. degree in computer science from the Peking University in 1985, and the M.S. and Ph.D. degrees in computer science from the University of Illinois at Urbana-Champaign in 1987 to 1990, respectively.

Currently, he is an assistant professor in the Computer Science Department of the University of California, Los Angeles. From 1986 to 1990, he was a research assistant in the Computer Science Department of the University of Illinois. He worked at the Xerox Palo Alto Research Center in the summer of 1987. He worked at the National Semiconductor Corporation in the summer of 1988. His research interests include computer-aided design of VLSI circuits, fault-tolerant design of VLSI systems, and design and analysis of efficient combinatorial and geometric algorithms.

Dr. Cong received the Best Graduate Award from the Peking University in 1985. He was awarded a DEC Computer Science Fellowship in 1988. He received the Ross J. Martin Award for excellence in research from the University of Illinois at Urbana-Champaign in 1989. He received the National Science Foundation Engineering Research Initiation Award in 1991.