NEMO: A New Implicit-Connection-Graph-Based Gridless Router With Multilayer Planes and Pseudo Tile Propagation

Yih-Lang Li, *Member, IEEE*, Hsin-Yu Chen, and Chih-Ta Lin

***Abstract*—The implicit-connection-graph-based router is superior to the tile-based router in terms of routing graph construction and point querying. However, the implicit connection graph has a higher degree of routing graph complexity. In this paper, a new multilayer implicit-connection-graph-based gridless router called NEMO is developed. Unlike the first implicit-connection–graph-based router that embeds all routing layers onto a routing plane, NEMO constructs a routing plane for each routing layer. Additionally, each routing plane comprises tiles, not an array of grid points with their connecting edges, and consequently, the complexity of the routing problem decreases. Each grid point then represents exactly one tile or its left-bottom corner such that a tile query is equivalent to any point query inside the queried tile, and a grid maze becomes tile propagation. Furthermore, to accelerate path search, continuous space tiles are combined as a pseudo maximum horizontally or vertically stripped tile. Experimental results reveal that NEMO conducts a point-to-point path search around ten times faster than the implicit-connection-graph-based router. General-purpose routing by NEMO also improves routing performance by approximately 1.69×–55.82×, as compared to previously published works based on a set of commonly used MCNC benchmark circuits.**

***Index Terms*—Physical design, routing.**

I. Introduction

In the era of deep-submicrometer (DSM) technology and system-on-chip design methodology, very large-scale integrated (VLSI) designs engender challenges in optimizing layouts resulting from ongoing reductions in device size, wire width, and wire space. Interconnection optimization is crucial in minimizing delay and noise and optimizing reliability of modern chip designs. Wire sizing and wire spacing have been proposed as techniques in optimizing interconnectivity, thus imposing variable-width and variable-space constraints on detailed routers. Variable-rule routing raises the requirement for a router other than a uniform-grid router. Nonuniform-grid routers and tile-based routers, which are also called gridless routers, are commonly employed in variable-rule routing [1]–[12]. To accommodate variable-rule routing, gridless routers require more complex data structures than do uniform-grid routers to construct routing graphs quickly, as well as to verify the design rule check legality of a move on a routing graph. In the meantime, modern designs have substantially increased the instance size of the routing problem to motivate novel routing techniques and frameworks for high-performance gridless routing. Among these designs, multilevel design framework [13]–[18] and routing graph reduction [20] have been extensively investigated and have been successfully adopted to improve the routing performance.

Manuscript received May 2, 2006; revised August 18, 2006. This work was supported by the National Science Council of Taiwan under Grants NSC 94-2220-E-009-043, NSC 95-2220-E-009-019, and NSC 95-2220-E-009-026. This paper was recommended by Guest Editor P. H. Madden.

Y.-L. Li and C.-T. Lin are with National Chiao-Tung University, Hsinchu 300, Taiwan, R.O.C. (e-mail: ylli@cs.nctu.edu.tw; ctlin@cs.nctu.edu.tw).

H.-Y. Chen is with Faraday Technology Corporation, Hsinchu 300, Taiwan, R.O.C. (e-mail: yorke_ch@faraday-tech.com).

Digital Object Identifier 10.1109/TCAD.2007.891381

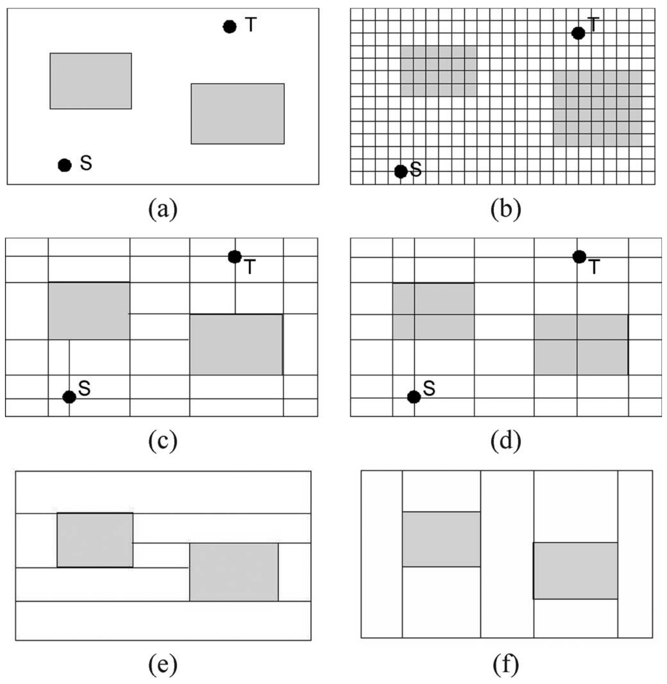

Fig. 1. (a) Layout with two obstacles and two terminals. (b) Fine-uniform-grid graph of the layout. (c) Connection graph of the layout. (d) Implicit connection graph of the layout. (e) MHS tile plane for the case in (a). (f) MVS tile plane.

A. Variable-Width and Variable-Space Routing

Based on the flexibility of dealing with variable wire widths and spacing rules, detailed routers can be categorized as either grid or gridless routers; the former search for paths on a uniform-grid graph, whereas the latter search nonuniform-grid graphs or nongrid graphs such as a tile-based graph. A gridless router can deal with variable-width and variable-space routing more efficiently than a grid router can.

Direct realization of a gridless router makes use of fine uniform grids or manufacturing grids. Fig. 1(a) presents a layout containing two obstacles with two routing terminals S

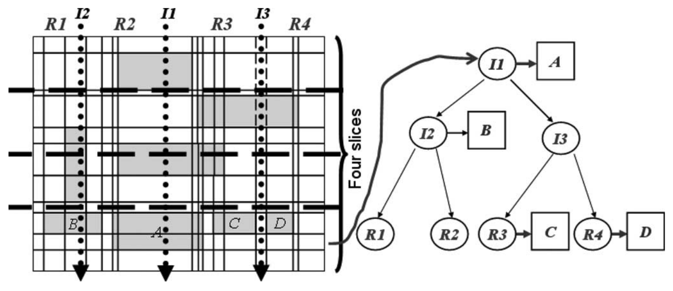

Fig. 2. Example of a slit and interval tree. The figure on the right is the interval tree of the bottom slice and is produced by three cut lines. Objects A and B are intersected with cut lines $I1$ and $I2$, respectively, and objects C and D are contained in the regions $R3$ and $R4$, respectively.

and T; Fig. 1(b) presents the routing graph, realizing the fine-uniform-grid model. The induced substantial routing graph for a large design requires such substantial computing and memory resources that fast path searching is infeasible. Different approaches have been explored to decrease the routing graph [1]–[12], of which the connection graph and tile-based graph are the most widely used. In [9], a connection graph is constructed by extending the boundary lines of all obstacles until reaching the boundary of other obstacles or the routing boundary [Fig. 1(c)]. The deficiency in this approach is the nonoptimal result in multilayer routing and the expensive cost for representing the connection graph. Cong *et al.* presented an implicit representation for a connection graph that allows borderlines of each expanded obstacle to penetrate any other obstacle until routing boundaries are reached [11], [12] [Fig. 1(d)]. The boundary lines of all expanded obstacles on all layers are integrated into a single routing plane. A grid point on an implicit connection graph may be located inside a blockage, and then routing on a newly visited grid point may not be legal. An efficient query structure, called a slit and interval tree, has been proposed for rapidly querying the legality of a grid point [11], [12]. The routing region is first partitioned into several slices, each with its own interval tree. Interval trees are generated by bisecting a sliced region. Fig. 2 depicts an interval tree induced by three cut lines. The cut line $I1$ corresponds to the root node of the interval tree, and the cut lines $I2$ and $I3$ correspond to two internal nodes in the second level of the interval tree. If an obstacle intersects a cut line, it is attached to the node of that cut line. For instance, cut lines $I1$ and $I2$ overlap objects A and B, respectively. Therefore, objects A and B are attached to the root node and the left internal node in the second level of the interval tree. Four leaf nodes correspond to four intervals, which are separated by the three cut lines, and all objects in an interval that do not intersect any of the three cut lines are attached to the leaf node of this interval. Although the implicit representation has more graph nodes than in [9] and time must be taken to check whether a move to an unvisited node is legal, optimal multilayer routing, fast querying of the legality for a move, and fast construction of an array in a nonuniform-grid graph are the primary contributions of that study [11], [12].

The other well-known gridless approach is the tile-based router [1]–[7] that partitions the entire routing region by existing obstacles into two tile types—space tiles and block tiles—and organizes all tiles using a corner-stitching data structure [21]. The routing tile plane of a horizontal/vertical routing layer is produced in a maximum horizontally/vertically stripped (MHS/MVS) property by extending the horizontal/vertical border lines of all obstacles until any obstacle or routing boundary is reached [Fig. 1(e) and (f)]. Both the implicit-connection-graph and tile-plane approaches can find an optimal path in point-to-point routing. Table I compares the time complexity analyses of four important operations of tile-based and implicit-connection-graph-based routers. The average number of visited nodes on the slit and interval trees in a point-finding operation is of the order of $\log(n_s \times n_i)$, where n_s is the number of slices used to partition the routing region, and n_i is the number of leaf nodes in the interval tree. The average number of visited tiles in a point-finding operation on a tile plane is of the order of $\sqrt{n_t}$, which is the mean number of tiles in a row or a column for a tile plane that contains n_1 tiles. Neighbors can be found by both routers in constant time. The time complexity of checking the legality of an unvisited grid point on an implicit connection graph is $\mathrm{O}(n_o \times \log(n_i))$, where n_o is the average number of objects attached to a node in the interval tree, whereas the tile-based router requires only to check the type of a tile. n_o can be held constant to reduce the complexity of the legality check at the expense of increasing the need for more memory to store the interval trees. Then, the time complexity of the legality check for an implicit connection graph is $\mathrm{O}(\log(n_i))$. The relationship between n_s and n_o can be formulated as

$$n_o = \left\lfloor \frac{H}{n_s \times \text{pitch}} \right\rfloor + 1 \tag{1}$$

where H is the height of the routing region, and pitch is the routing pitch for this routing plane.

For single-object insertion, an implicit-connection-graph-based router requires an array reconstruction operation with a time complexity of $\mathrm{O}(n)$, where n is the average number of objects in the point array; for the tile-based router, an object must be inserted with a time complexity of $\mathrm{O}(\sqrt{n_t} + m^2)$, where m is the average number of objects that overlap the inserted tile. The former and latter terms refer to locating the starting tile and beginning the splitting and merging of tiles on a tile plane. The implicit connection graph outperforms the corner-stitching plane in the querying operation, whereas the corner-stitching tile plane has a substantial advantage over the implicit connection graph in the insertion of a short wire, which results in a small value of m.

TABLE I
TIME COMPLEXITY ANALYSIS OF FIVE IMPORTANT OPERATIONS FOR THE ROUTING GRAPHS OF THE TILE PLANE AND IMPLICIT-CONNECTION-GRAPH-BASED ROUTERS

	Point finding	Neighbor finding	Legality check	Single object insertion
Implicit connection graph	*log* $(n_s \times n_i)$	*constant time*	$n_o log(n_i)$	N
Tile plane	$\sqrt{n_t}$	*constant time*	*constant time*	$\sqrt{n_t} + m^2$

B. Multilevel Design Framework for Routing

A new routing framework, multilevel design framework, was introduced to enhance significantly the performance of a routing system [13]–[18]. In the work of Cong *et al.* [13], [14], a multilevel global routing was first proposed. After coarsening the tiling routing region and estimating the routing resource, the initial routing result is generated utilizing a multicommodity flow algorithm in the coarsest level and a constrained maze algorithm to refine the routing result for solving local congestion during the uncoarsening stage. A gridless detailed router then determines detailed routing paths for all nets following the search region constraint, which consists of tile-to-tile global paths for each net. In [15], global routing and detailed routing algorithms are first fully integrated into a multilevel framework, and a net is routed when it becomes a local net during the coarsening stage, in which the local net concept is defined using a different area-partitioning scale ranging from a finer to a coarser level. Failed nets and routing solution refinement are completed during uncoarsening. Chen *et al.* proposed a two-pass bottom-up routing framework, which resembles a multilevel framework, considering double-via insertion [19]. Global routing and detailed routing are performed level by level in each coarsening stage, enabling local nets to be processed first for routability and via optimization. The routing framework in [19] outperforms all multilevel gridless routers.

C. Simplification of Routing Graph

Another strategy for meeting the challenge of the increasing size of the circuit is to develop a new or enhanced routing algorithm and routing model. For the tile-based router, routing graph reduction approaches, which involve redundant tile removal and neighboring-tile alignment, have been discussed [20]. Redundant tile removal changes to block tiles two types of space tiles, namely: 1) one-conjunct tiles and 2) zero-conjunct tiles, which have one and zero entry points from the neighboring space tile, respectively, and then provides no exit for further propagation. The tile-based routing plane is then streamlined. Neighboring-tile alignment follows shrinking rules to shrink space tiles in an attempt to merge top/left and bottom/right neighboring tiles on a horizontal/vertical routing plane without reducing routability. Routing graph reduction guarantees the production of an optimal path if each routing layer has a unique cost in each routing direction; a halving of runtime in tile propagation is achieved.

Although the implicit-connection-graph-based router has the principal advantages of fast routing graph construction and query operation, embedding all layer obstacles in a routing plane requires a massive routing graph for a large design. Consider a two-layer routing: many horizontal gridlines are induced by existing expanded horizontal wires; similarly, many vertical gridlines are induced by existing expanded vertical wires on a vertical routing plane. If two routing planes are merged, then a horizontal gridline, for instance, which is induced by an expanded horizontal wire and does not overlap any border of all of the expanded vertical wires, will yield numerous grid points by crossing the vertical borders of all of the expanded vertical wires. Fig. 3 displays an implicit connection graph for two-layer

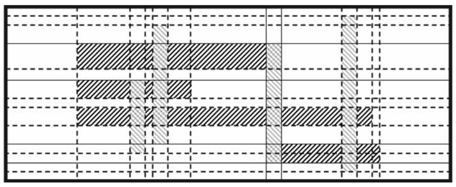
Fig. 3. Implicit connection graph is produced by embedding a horizontal and a vertical routing plane into a single routing plane. Each bolded gridline is obtained from the borderlines of some expanded obstacles in a routing layer. The complexity of an implicit connection graph increases as the number of bolded gridline increases.

routing; each dotted gridline is obtained from only the borderlines of some expanded obstacles in the same routing layer, and intersects with all gridlines induced by the borderlines of those expanded obstacles on all orthogonal layers. Accordingly, an implicit connection graph tends to be large. For example, in case $D2$, in which a chip design is employed in point-to-point routing experiments, 1 650 704 metal-2 rectangles and 1 018 211 metal-3 rectangles are involved. The implicit connection routing graph for this design comprises 536 214 892 grid points, and no fast path search can be expected to be performed on such a large routing graph. Furthermore, the limitation on array size also inhibits the application of the implicit-connection-graph-based router. A single routing-plane scheme suffers from unnecessary moving for a routing with at least three routing layers. Since the routing pitch of a high routing layer generally exceeds that of a low routing layer, for example, the obstacle borders on two horizontal routing layers will probably not align with each other, leading to unnecessary queries regarding contiguous grid points on different nonadjacent layers. In [14], a distributed grid graph has been proposed to simplify the connection graph. A 2-D point array of the entire routing plane is distributed into global cells (GCells), each of which consists of its own 2-D point array, and the connection graph for a GCell is influenced only by those obstacles that overlap the GCell. In the connection graph of a global path, inside which the detailed routing propagates, borders of the obstacles inside a GCell will not reach the region of neighboring GCells, except in that the global path includes the GCell and its neighbors.

D. Paper Contributions and Organization

This paper proposes a new and much-simplified implicit-connection-graph-based router that combines fast pseudo tile propagation and enhanced querying for a slit and interval tree during point-to-point routing to result in a significant drop in runtime. Herein, a novel implicit-connection-graph-based gridless router comprises a multiplane implicit connection graph, a nonzero-width wire model, a projection array (PA), and a pseudo MHS or MVS tile (PMT) extraction and propagation technique. The connection graph of the proposed new implicit-connection-graph-based router is of a similar scale to that of the tiling routing plane, and the same constant-time query is used for layer switching as that used in the implicit-connection-graph-based router. Then, a point-to-point gridless router can optimally run almost ten times faster than that in [11]. Based on the proposed approach, a full-chip gridless router that outperforms all well-known multilevel gridless routers, as well as

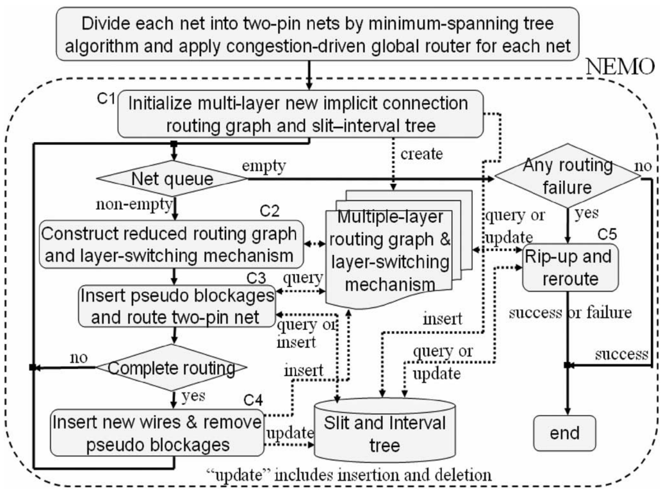
Divide each net into two-pin nets by minimum-spanning tree algorithm and apply congestion-driven global router for each net
NEMO
C1 Initialize multi-layer new implicit connection routing graph and slit–interval tree
create
Net queue
empty
non-empty
Any routing failure
no
yes
C2 Construct reduced routing graph and layer-switching mechanism
Multiple-layer routing graph & layer-switching mechanism
query or update
C5 Rip-up and reroute
C3 Insert pseudo blockages and route two-pin net
query
query or insert
insert
success or failure
no
Complete routing
yes
query or update
success
insert
C4 Insert new wires & remove pseudo blockages
update
Slit and Interval tree
end
"update" includes insertion and deletion

Fig. 4. NEMO design flow.

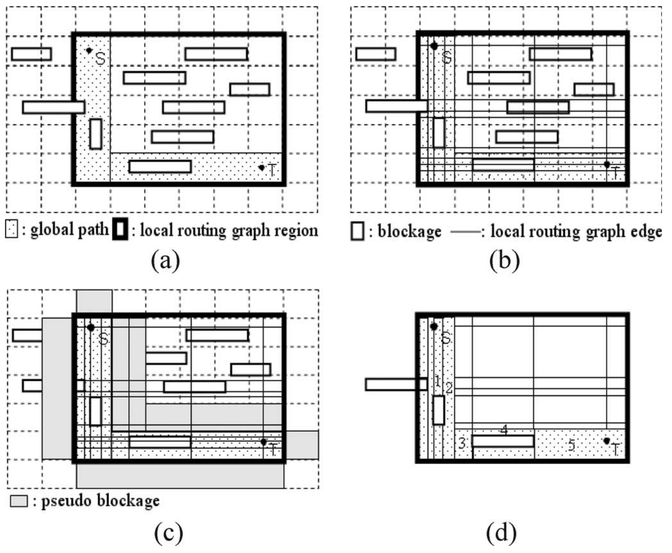
S
T
▢: global path ▮: local routing graph region
(a)
▢: blockage —: local routing graph edge
(b)
▢: pseudo blockage
(c)
1
2
3
4
5
(d)

Fig. 5. (a) Single-layer two-pin routing example guided by an L-shaped global path. (b) Reduced local routing graph established based on all obstacles overlapping the global path. (c) Pseudoblockages inserted around the global path. (d) Path (tiles $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5$) found by connecting a series of pseudo MVS space tiles.

a multilevel grid-based router that outperforms in routing speed and routing quality, was also developed.

The rest of this paper is organized as follows. Section II outlines the design flow of NEMO. Section III presents the proposed routing model, including the multiplane routing graph, the nonzero-width wire model, and the efficient layer-switching mechanism. Section IV presents the novel implicit-connection-graph-based point-to-point routing scheme. Section V elucidates approaches for implementing the full-chip gridless router. Section VI presents the experimental results. Finally, Section VII draws conclusions.

II. NEMO Overview

In this paper, a fast gridless detailed router, called NEMO, is designed; hence, the full-chip routing is completed only with global routing followed by detailed routing. The routing area is first partitioned into several equal-sized GCells, and the minimum spanning tree (MST) algorithm is applied to each net to build its routing topology. This routing topology decomposes each net into several two-pin routings. That is, each point-to-point routing corresponds to an edge of the MST. A congestion-driven global routing then determines the global path for each two-pin connection. This global path comprises a list of connected tiles. Finally, NEMO completes the detailed routing.

The main features of NEMO for promoting routing performance comprise a new implicit-connection-graph-based routing model $(F1)$, PMT extraction and propagation $(F2)$, gridline reduction $(F3)$, and pseudoblockage insertion $(F4)$. Feature $F1$ enables the router, which is based on an implicit connection graph, to behave as a tile-based router with a constant-time layer-switching capability; feature $F2$ enhances the queries on the slit and interval tree as well as path searching on the routing graph; feature $F3$ constructs a simplified routing graph for point-to-point routing; and feature $F4$ decreases the pseudo tile extraction time and constrains detailed routing to the global path.

Fig. 4 illustrates the NEMO design flow. NEMO first initializes the new global multiple-layer connection routing graph (chip area) and the slit and interval tree $(C1)$. NEMO builds a simplified local routing graph in the range of the bounding box of two routed pins through gridline reduction for current two-pin routing. A constant-time layer-switching data structure based on the simplified local routing graph is then established $(C2)$. Pseudoblockages are then inserted around the global path, and routing is conducted $(C3)$. If the current two-pin routing is complete, the newly generated wire segments are inserted into the global routing graph and the slit and interval tree, and pseudoblockages for recently completed routing are eliminated from the slit and interval tree $(C4)$. If the previous stage has any failed nets, then NEMO resolves the incomplete routings with a rip-up and reroute process $(C5)$. The contributions of this paper are in the components $C1$, $C2$, and $C3$. Fig. 5 illustrates a $C2{-}C3$ example of one-layer two-pin routing guided by an L-shaped global path, where the GCells are filled with a dotted pattern. The rectangle with bold borders in Fig. 5(a) shows the range of a simplified local routing graph. The local routing graph is rectangular instead of L-shaped since an array must be built as a rectangular structure. Fig. 5(b) depicts the reduced local routing graph derived from the borders of those obstacles overlapping the L-shaped global path. The reduced routing graph is fairly simple, even though the two-pin bounding box contains many obstacles. The following two-pin routing is conducted on the local routing graph instead of on the global routing graph. Before starting two-pin routing, pseudoblockages are inserted into the slit and interval tree to isolate the global path such that no extracted pseudo space tiles fall outside the global path. Fig. 5(c) shows the influence of pseudoblockages. The grid maze is replaced by pseudo tile propagation during routing. Fig. 5(d) displays all pseudo MVS space tiles and one possible routing path, which starts at tile 1, passes through tiles 2, 3, and 4, and finally reaches target tile 5. Other works provide detailed tile propagation examples [4], [7].

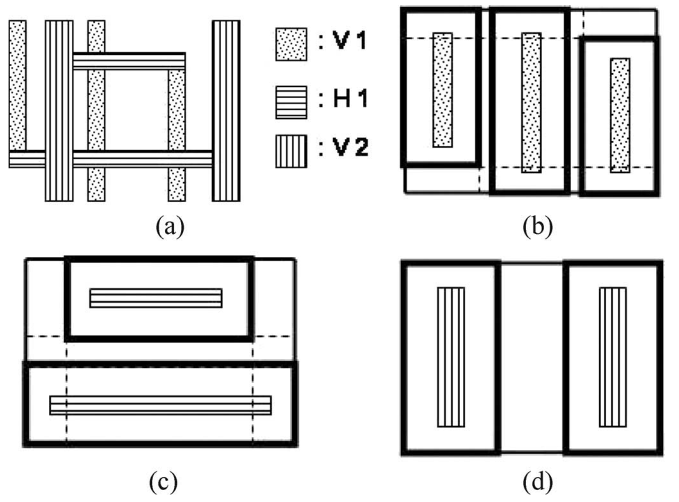

Fig. 6. (a) Example of three-layer routing. (b) Routing plane for the $V1$ layer. (c) Routing plane for the $H1$ layer. (d) Routing plane for the $V2$ layer.

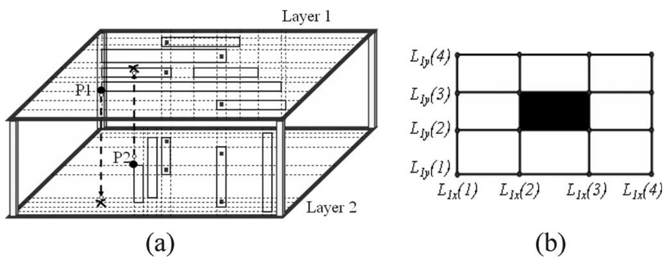

Fig. 7. (a) Grid misalignment example. Grid point P1(P2) has no adjacent grid point at the same position on the adjacent layer L2(L1). (b) Routing plane considered to be a tile plane. The ith vertical gridline on layer 1 is denoted as $L_{1x}(i)$.

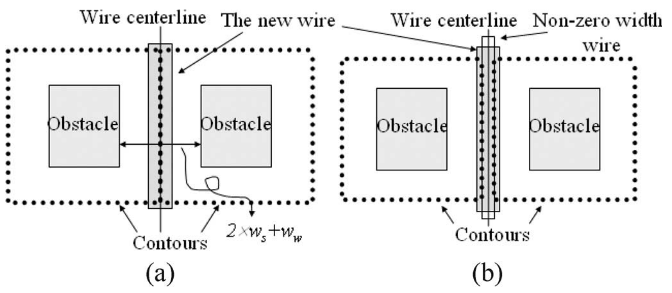

Fig. 8. Example of two obstacles separated by a distance of $2 \times w_s + w_w$. (a) Two expanded obstacles abut on one side for the zero-width wire model. (b) Space tile of width of two times of v_s that exists between two expanded obstacles for the nonzero-width wire model.

III. Routing Model

A. Multiplane Routing Graph

The implicit representation for the connection graph in [11] stores in two arrays the sorted vertical and horizontal grid lines generated by the boundaries of all expanded obstacles of all routing layers. All obstacles are expanded in a size such that the centerline of a new path can be placed along the borders of the expanded obstacles and no design rule violation will occur. This approach ensures that an optimal path is located; however, the principal problem is that the underlying routing graph tends to be too large for fast path searching. This difficulty is primarily caused by the fact that different layers of the same preferred routing direction frequently are subject to different space and width rules such that the boundaries of the expanded obstacles of these layers probably do not match. To resolve this problem, a new implicit connection graph is constructed as a multiplane graph in which each plane is responsible for one routing layer and stores all expanded obstacles of that layer. Thus, a routing problem of n routing layers contains n routing planes, and the boundaries of an expanded obstacle are not superimposed on the routing planes of other layers. Fig. 6(a) presents a three-layer routing; Fig. 6(b)–(d) shows its associated three routing planes.

An earlier work [11] regarded the connection graph as an array of grid points and connected edges and, consequently, completed routing by mazing over these nodes. After a routing plane was configured into multiple routing planes, regarding a routing plane as an array of grid points is inadequate since a grid point on a layer likely has no adjacent grid point at the same position on an adjacent layer, so the grids on adjacent layers are misaligned. Fig. 7(a) shows an example of grid misalignment at two points $P1$ and $P2$. Routing from layer 1/2 to layer 2/1 at the point $P1/P2$ is not feasible. Layer switching then becomes unnatural and somewhat complex under this scenario. If a routing plane is regarded as comprising tiles, each of which is identified by its left-bottom corner, then each grid point exactly represents one tile. The layer-switching problem caused by misalignment of grids on adjacent layers can be solved by layer switching on two overlapping tiles of adjacent layers and postponing the determination of the exact via position. Another benefit of the tile concept is that the tiles must be fewer than the grid points. Therefore, the implicit connection graph herein is considered to be a tiling structure rather than a grid array. The ith vertical/horizontal grid line in the implicit connection graph on layer 1 is denoted by $L_{lx}(i)/L_{ly}(i)$ herein. Fig. 7(b) presents a routing plane containing an expanded obstacle and its grid line notation. The black tile is derived as $T(L_{1x}(2), L_{1y}(2))$.

B. Nonzero-Width Wire Model

Path searching for a gridless router can be a zero-width or nonzero-width wire model. In the zero-width wire model, an obstacle i is expanded by a contour of width $w_s + w_w/2$, say ew_i, where w_w is the wire width of the routed net, and w_s is the spacing rule for the routed net to obstacle i. In the nonzero-width wire model, the expanded region of $w_s + w_w/2$ is shrunk by a value v_s, such as a unit of width. For both models, the identified position is the centerline of a path.

The primary difference between these two wire models occurs when two obstacles are separated by a distance of $2 \times w_s + w_w$. The contours inserted into the two obstacles will abut on one side for the zero-width wire model, whereas the two contours will be separated by a distance of two times of v_s for the nonzero-width wire model [Fig. 8(b)]. The work in [11] adopted a zero-width wire model since the routing plane was regarded as an array of grid points; the path search can then maze across the abutting line of two contours [Fig. 8(a)]. Conversely, since this study views a routing plane as a group of tiles, applying the zero-width wire model would delete an available routing path for the case presented in Fig. 8(a),

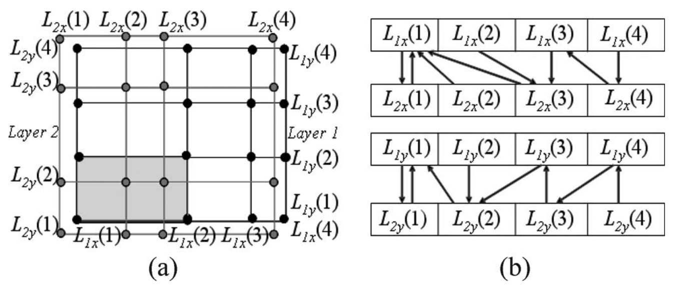
$L_{2x}(1)$ $L_{2x}(2)$ $L_{2x}(3)$ $L_{2x}(4)$
$L_{2y}(4)$ $L_{2y}(3)$ $L_{2y}(2)$ $L_{2y}(1)$
$L_{1y}(4)$ $L_{1y}(3)$ $L_{1y}(2)$ $L_{1y}(1)$
$L_{1x}(1)$ $L_{1x}(2)$ $L_{1x}(3)$ $L_{1x}(4)$
Layer 2
Layer 1
(a)
$L_{1x}(1)$ $L_{1x}(2)$ $L_{1x}(3)$ $L_{1x}(4)$
$L_{2x}(1)$ $L_{2x}(2)$ $L_{2x}(3)$ $L_{2x}(4)$
$L_{1y}(1)$ $L_{1y}(2)$ $L_{1y}(3)$ $L_{1y}(4)$
$L_{2y}(1)$ $L_{2y}(2)$ $L_{2y}(3)$ $L_{2y}(4)$
(b)

Fig. 9. (a) New implicit connection graph for two-layer routing with two obstacles. A projection from *Layer 1* to *Layer 2* identifies the reachable tiles on *Layer 2* from tile $T(L_{1x}(1), L_{1y}(1))$ on *Layer 1*. (b) Example of two adjacent *CA*s and their related *PA*s.

because two contours abut with no space tile in between. The nonzero-width wire model can leave a space tile of width of two times of v_s. Thus, the nonzero-width wire model is adopted in this paper.

C. Multilayer Model

The grid lines of all layers are embedded in a single plane in [11]; consequently, a layer-switching point between adjacent layers can be determined in constant time. Fast layer switching is extremely important for an effective gridless router. To accelerate layer switching on the proposed implicit connection graph, additional arrays, called *PA*s, are used to record the first reachable tiles on layers adjacent to each tile. Two arrays are then available: a coordinate array (*CA*) and a *PA*. Each *CA* stands for the left-bottom corner of a tile, and each *PA* signifies the first reachable tiles from one tile on adjacent layers. From the description above, the formal definition is stated as follows: The ith element of a *CA* for layer l, say $L_{lx}(i)$, has a related element in the *PA*, and it points to the jth element in the *CA* of adjacent layer, say $l+1$, when $L_{(l+1)x}(j) \leq L_{lx}(i) < (L_{(l+1)x}(j+1)$. Notation of the ith horizontal element of the *PA* from layer l to $l+1$ is $PA_{(l,l+1)x}(i)$.

For instance, consider the case in Fig. 9(a) with two blockage tiles $T(L_{1x}(2), L_{1y}(2))$ and $T(L_{2x}(2), L_{2y}(2))$. If the tile propagation intends to switch from a Layer 1 tile $T(L_{1x}(1), L_{1y}(1))$ to a Layer 2 tile, then the query concerning reachable tiles on Layer 2 must be computed by projecting the region of tile $T(L_{1x}(1), L_{1y}(1))$ on Layer 1 onto Layer 2. All tiles on Layer 2 covering the projection region are the reachable tiles on Layer 2 through the tile $T(L_{1x}(1), L_{1y}(1))$. Through the *PA*, the project region can then be figured out using the tile index of Layer 2 in constant time. Fig. 9(b) depicts the assignment of the *PA*s on horizontal and vertical directions for layers 1 and 2. For instance, $L_{2x}(3) < L_{1x}(2) < L_{2x}(4)$ and $L_{1x}(1) L_{2x}(2), L_{2x}(3) < L_{1x}(2)$, so $PA_{(1,2)x}(2)$ equals 3, and $PA_{(2,1)x}(2)$ and $PA_{(2,1)x}(3)$ equal 1. To identify which tiles on adjacent layers overlap the current tile, say $T(L_{1x}(1), L_{1y}(1))$, $PA_{(1,2)x}(1)$ and $PA_{(1,2)x}(2)$ can immediately infer the horizontal range of reachable tiles on Layer 2, starting from $L_{2x}(1)$ to $L_{2x}(3)$. Similarly, the vertical range of reachable tiles on Layer 2 from $L_{2y}(1)$ to $L_{2y}(2)$ can be quickly identified through $PA_{(1,2)y}(1)$ and $PA_{(1,2)y}(2)$. Finally, the tiles located inside the area enclosed by the horizontal and vertical ranges of reachable tiles are the tiles that tile $T(L_{1x}(1), L_{1y}(1))$ overlaps. In this case, tiles $T(L_{2x}(1), L_{2y}(1))$, $T(L_{2x}(1), L_{2y}(2))$, $T(L_{2x}(2), L_{2y}(1))$, $T(L_{2x}(2), L_{2y}(2))$, $T(L_{2x}(3), L_{2y}(1))$, and $T(L_{2x}(3), L_{2y}(2))$ are candidates for layer switching. Therefore, the query for layer switching utilizing the proposed scheme also remains constant. On the other hand, construction of a *PA* does not require additional computation time since the required number of comparisons for constructing the *PA*s of two adjacent *CA*s is the same as that needed to merge two *CA*s into a single *CA* for constructing a unified routing plane for the traditional implicit-connection-graph-based router.

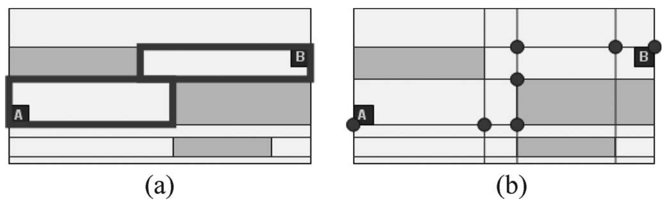

Fig. 10. Routing example with three obstacles. (a) Shortest path search from A to B on an MHS plane that requires only two propagation steps. (b) Six steps required for the same shortest path search on a grid plane.

IV. New Implicit-Connection-Graph-Based Point-to-Point Routing

Implicit-connection-graph-based routing is superior to tile-based routing when constructing a routing graph. However, tile-based routing partitions a routing region to obtain the MHS and MVS tiles such that the expansion over a tile likely advances more farther than implicit-connection-graph-based routing. Fig. 10 displays a routing example with three obstacles. Shortest path search from A to B on an MHS plane only requires two propagation steps, as shown in Fig. 10(a); six steps are required for the same shortest path search on a grid plane. Further grouping the tiles on the implicit connection graph to form PMTs is inspired by the tile propagation scheme of tile-based routing. Through the PMT propagation, NEMO acquires considerable acceleration in routing speed. The following illustrates the method utilized to generate the PMT and propagate PMTs on the new implicit connection graph.

A. Tile Query

In this paper, the slit tree and interval tree data structures, which were reported in [11], are also employed to store existing paths for querying the legality of each move. For a horizontal routing layer, the plane is first split horizontally into several equal-width slices. All obstacles overlapping a slice are stored in an interval tree that is produced by imposing uniform vertical lines on that slice. During path searching, a legality assessment for an unvisited tile is the same as that for a grid point. The legality check for a tile, say $T(L_{nx}(i), L_{ny}(i))$, is performing the legality check for any point inside a tile—for instance, $(L_{nx}(i)+1, L_{ny}(i)+1)$. The legality check starts from the root of the interval tree of the slice whose vertical range contains the queried point and traverses the tree until a leaf node is reached. If the queried point is not enclosed by any traversed rectangles, the point is considered free, and the tile containing the queried point is a space tile.

Since the slit and interval tree stores unexpanded obstacles to prevent frequent updating of the tree structure owing to the change of design rules, the local search around the queried point must search for all rectangles within the largest expanded distance $\max_i ew_i$ to the point for design rule verification. If the point, which is located in the middle of a tile, is selected for the legality check, the number of obstacles with which the point will be compared can be reduced. If the distance from the four sides of a tile to its center equals $\max_i ew_i$, then the local search is simplified to find the obstacles overlapping the middle point.

B. PMT Extraction

Path search on the new connection graph has increased efficiency when adjacent space tiles are organized as a PMT. If a point query is found to be within any blockage, then the query halts and the blocked tile is returned; otherwise, once a tile is confirmed to be a space tile, the traversal of the interval tree continues to identify the closest obstacles on the left and the right, as well as the top and the bottom, of the space tile for MHS and MVS tile extraction, respectively. For instance, when the legality check for a tile is completed in a horizontal routing layer, the traversal of the tree halts at a leaf node. PMT extraction proceeds following the legality check. During the process of legality check, the closest obstacles, whose vertical ranges contain the y-coordinate of the queried point, to the queried point are recorded. After the legality check, the borders of two sides of the MHS tile are identified if two closest obstacles to the queried point are stored on two sides; otherwise, the traversal of the tree proceeds toward an undetermined border to visit the leaf nodes and thus acquire the border of the undetermined side. Whenever the parent of a currently visited leaf node has not yet been visited during extraction, the traversal of the tree proceeds upward to determine whether other obstacles are present. The upward traversal is necessary since a wire will be stored at an internal node of an interval tree if it intersects any cut line. A long wire is probably stored at a high-level internal node of an interval tree; therefore, the upward traversal searches for those wires stored on high-level unvisited internal nodes and has to terminate on the node whose parent has been visited. The following operation for identifying the level of the closest common internal node to two leaf nodes is introduced to analyze the required number of visited internal nodes in an upward traversal. Fig. 11 displays an interval tree with n leaf nodes and $\log_2 n$-level internal nodes; the basic properties of a balanced binary tree can thus be exploited. The root node is at level 1, and each internal node has a level number that is one larger than its parent, as revealed in Fig. 11. A *maximum-level common ancestor* (MLCA) of two leaf nodes is the common ancestor of two leaf nodes closest to the leaf nodes; the level of the MLCA of leaf nodes i and j is defined as

$$\bigcap_{\text{ancestor}}^{\max}(i,j) = \max\left\{k \middle| k \in \text{level number} \ni \left\lfloor \frac{i}{2^t} \right\rfloor = \left\lfloor \frac{j}{2^t} \right\rfloor\right\} \tag{2}$$

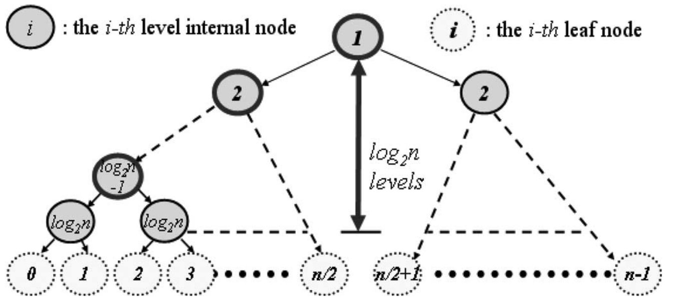

Fig. 11. Interval tree with n leaf nodes and $\log_2 n$-level internal nodes. The internal nodes outlined by bold borders are the common ancestors of leaf nodes 0, 1, 2, and 3.

where t denotes the distance from level k to the leaf node ($t = \log_2 n - k + 1$). For instance, in Fig. 11, leaf nodes 0, 1, 2, and 3 have no common ancestor on level $\log_2 n(\lfloor 1/2^1 \rfloor \neq \lfloor 2/2^1 \rfloor)$, and their common ancestor closest to them, or the MLCA, is on level $(\log_2 n) - 1(\lfloor 0/2^2 \rfloor = \lfloor 1/2^2 \rfloor = \lfloor 2/2^2 \rfloor = \lfloor 3/2^2 \rfloor)$. The other common ancestors are outlined with bold circles in Fig. 11. When a horizontal move is made on leaf nodes, say from node i to node j, the internal nodes from node j's parent to the internal node next to the MLCA of nodes i and j (MLCA's child node) are the unvisited nodes and have to be traversed by the upward traversal that starts at node j. A traversal of the internal nodes from the MLCA of nodes i and j to the root has been performed previously and is thus unnecessary. For example, the MCLA of leaf nodes 2 and 3 is their parent node, so no unvisited nodes exist for a move from leaf node 3 to leaf node 2. The MCLA for a move from leaf node 2 to leaf node 1 is on level $\log_2 n - 1$, which means that the unvisited node is the parent of leaf node 1. The number of unnecessary internal nodes to be traversed during PMT extraction determines the efficiency of the proposed extraction scheme. For a leftward/rightward move from leaf node i to leaf node $i - 1/i + 1$, if i is odd/even, such that the MLCA of nodes i and $i - 1/i + 1$ is their parent and is on level $\log_2 n$ (e.g., a leftward/rightward move from leaf node 3/2 to leaf node 2/3 in Fig. 11), then the parent of node $i - 1/i + 1$ must have been visited during the upward traversal that starts at node i and an upward traversal is unnecessary; otherwise, a path of $(\log_2 n - \bigcap_{\text{ancestor}}^{\max}(i, i - 1/i + 1))$ internal ancestor nodes from level $\log_2 n$ to level $(\bigcap_{\text{ancestor}}^{\max}(i, i - 1/i + 1) + 1)$ must be traversed.

Lemma 1: The identification of the left border of a PMT starts with leftward traversal at the leaf node m and ends at leaf node n, where p even leaf nodes and q odd leaf nodes are to be visited in $m - n$ horizontal moves $(p + q = m - n)$, and the q odd leaf nodes are the nodes $n_{o1}, n_{o2}, \ldots,$ and n_{oq}. The number of unnecessary internal nodes to be traversed during the identification of the left border is $p \times \log_2 n + \sum_{i=n_{o1}}^{n_{oq}} \bigcap_{\text{ancestor}}^{\max}(i, i + 1)$.

Proof: p moves are made from an odd leaf node to an even leaf node and q moves are made from an even leaf node to an odd leaf node during left-border identification. Whenever a move is from an odd leaf node to an even leaf node, upward traversal is unnecessary, and the advantage of not visiting $\log_2 n$ internal nodes is gained. p even nodes can

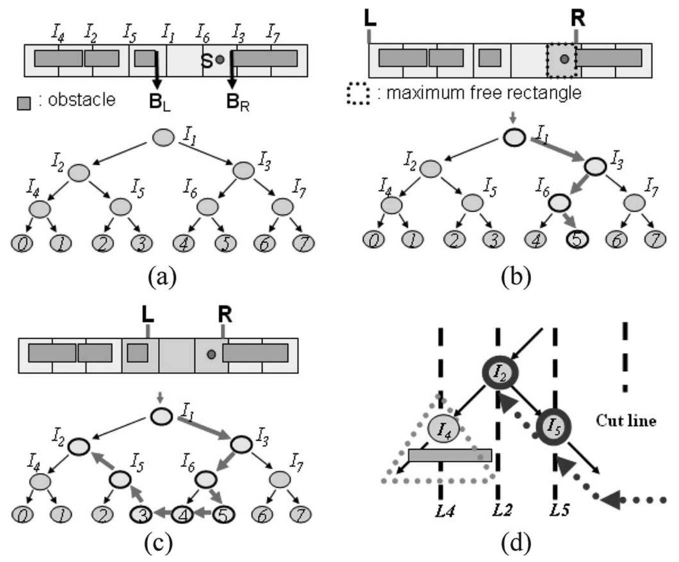

Fig. 12. Example of PMT extraction. (a) Interval tree of a slice. (b) Legality check process for a point that starts at the root and stops at leaf node 5; the right border of the PMT is defined after the legality check. (c) Leftward traversal of tree initiated to identify the PMT's left border and upward traversal performed if the current leaf node's parent has not yet been visited; the left-border identification process terminates at internal node I_2. (d) Left-border identification that ends on internal node I_2; any of the obstacles that are attached to node I_2 and its right subtree can determine the left border.

eliminate $p \times \log_2 n$ internal nodes for the traversal of the tree. However, whenever an odd leaf node, say i, is visited, its $\log_2 n - \bigcap_{\text{ancestor}}^{\max}(i, i+1)$ ancestors have to be traversed, and $\bigcap_{\text{ancestor}}^{\max}(i, i+1)$ internal nodes can then be eliminated for the traversal of the tree.

Similarly, the number of unnecessary internal nodes to be traversed for identifying the right border of a PMT can be deduced.

Lemma 2: The identification of the right border of a PMT starts with the rightward traversal at leaf node m and stops at leaf node n, where p odd leaf nodes and q even leaf nodes are visited in $m - n$ horizontal moves $(p + q = m - n)$, and the q even leaf nodes are the nodes $n_{e1}, n_{e2}, \ldots,$ and n_{eq}. The number of unnecessary internal nodes to be traversed during the identification of the right border is $p \times \log_2 n + \sum_{i=n_{e1}}^{n_{eq}} \bigcap_{\text{ancestor}}^{\max}(i, i-1)$.

Fig. 12 shows an example of interval tree traversal for PMT extraction. In this case, the check of legality of point S is initiated at the root and ends at leaf node 5, whose interval contains point S [Fig. 12(b)]; simultaneously, the right border of the PMT B_R has been determined. The identification of the left border for PMT extraction starts traversing toward the left. Since leaf node 4 contains no obstacle and its parent has been visited, the traversal moves to its left node, leaf node 3, which includes the first obstacles in the leaf node. Since its ancestors have not yet been visited, the traversal then proceeds upward, finally ending at internal node I_2; the left border of the PMT B_L is then identified. Fig. 12(c) displays the process of identifying the left border during PMT extraction.

Lemma 3: If a new border is located during an upward traversal, then the new border is identified, and the extraction of the border terminates.

Proof: For a given internal node, in the identification of the left border of a PMT, an obstacle that is stored in the left subtree of the internal node must be farther from the queried point than those obstacles that are stored at the internal node or its right subtree; similarly, the obstacles stored at the remaining leaf nodes left to the last visited leaf node must be far from the queried point. For instance, Fig. 12(d) shows three internal nodes, I_2, I_4, and I_5; in addition, Fig. 12(c) shows their associated cut lines. An upward traversal locates a PMT's left border at node I_2; no obstacle attached to the subtree rooted at node I_4 can then be found closer to the queried point than the obstacle on node I_2, because any obstacle, say O_c, attached to the subtree rooted at node I_4 must not intersect the cut line $L2$; otherwise, the obstacle O_c would be attached to node I_2 rather than to the subtree rooted at I_4.

If the upward traversal does not find a new border of the PMT, then the traversal for the unvisited leaf nodes and their ancestors continues until a new border is located or the routing-region boundary is reached. The two borders that are located in the traversal of the interval tree then define a pseudo MHS space tile. The entire tree traversal is through the index of each node and is very fast since the interval tree is a balanced tree.

In [12], two caches are exploited to improve the efficiency of queries. The blockage cache stores a fixed number of most recently found blockages, and the space cache stores a fixed number of most recently extracted maximum free rectangles. Such a rectangle is defined as the largest free region that encloses the queried point, as presented in Fig. 12(b). The locality of point-finding operations can cause a hit in the caches rather than in the interval tree. The issue that this scheme raises is how to increase the hit ratio. During the search for the path, the number of possible routing paths in the priority queue easily exceeds millions. If the existing routing paths appear over a wide area (e.g., for a point-to-path or path-to-path routing), then the hit ratio will drop. A large maximum free rectangle comprises more grid points than a small maximum free rectangle and potentially causes more hits in the caches. Therefore, the slit tree that employs the cache scheme tends to have large slices. However, large slices increase the number of obstacles attached to a node and, thereby, the time required to scan the obstacles when a node is being visited. Instead, the proposed PMT extraction scheme exhibits a steady performance improvement and precise improvement analysis. Unlike the cache scheme, the proposed PMT extraction scheme prefers small slices because the height of an extracted PMT is probably equal to the width of a wire. Small slices correspond to the need for only a few scanned obstacles in a node by PMT extraction, and extraction is accelerated. However, small slices increase the memory used to store interval trees. Experiments conducted in this paper demonstrate that a slice size of under ten routing pitches offers favorable query performance with reasonable memory usage.

Fig. 13 displays a PMT extraction result for a vertical-layer routing plane. A routing plane of 23 space tiles is reduced to seven space tiles after PMT extraction. Notably, a PMT does not likely form after a PMT extraction process. For example, if the queried point is $P1$ (Fig. 13), the top and bottom borders for PMT extraction are located at Z and Y, respectively; however,

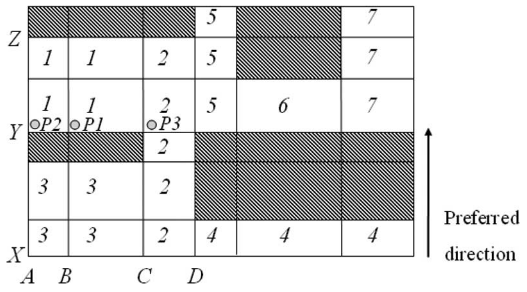

Fig. 13. PMT extraction can reduce the number of space tiles on this vertical-layer routing plane from 23 to 7. The labeling on each tile can be determined by performing PMT extraction on neighboring regions of currently processed tiles. The tiles of the same labels are grouped to form a PMT.

the tile bordered by the points (B, Y) and (C, Z) is not MVS. That is, for a vertical (horizontal)-layer routing plane, PMT extraction must be also conducted on the left and right (top and bottom) neighboring space tiles to determine whether the top and bottom (left and right) borders of the PMT are the same as those of the currently processed space tile after the first PMT extraction is completed. PMT extraction for PMT merging halts once the borders of a newly produced PMT are different from those of the currently processed space tile or a blockage is found. In Fig. 13, after the first PMT extraction on $P1$ acquires the top and bottom borders (Z and Y), PMT extractions on $P2$ and $P3$ are also performed on neighboring space tiles $T(A, Y)$ and $T(C, Y)$. Since the top and bottom borders for $T(A, Y)$ are also Z and Y, these two PMTs are integrated into one PMT, whereas another PMT, which ranges from (C, X) to (D, Z), is formed. For a routing plane, PMT extraction and merging can be repeatedly applied until all the space tiles of a routing plane have been processed. In Fig. 13, $P1$ is assumed to be the first queried point on the routing plane; the labeling number for each PMT is its generation order during PMT extraction. After PMT 1 is produced, PMT extractions are performed on tiles $T(B, X)$ and $T(D, X)$ to determine whether PMT 2 can merge with its neighbors. This scheme is simple and complete yet takes only a little time because not every tile is used in routing. One efficient approach is to classify each tile as one of four types—*unvisited*, *incomplete*, *free*, and *blocked*—and then to employ a 2-bit array to determine the status of each tile. Each tile is initially set to "unvisited." Whenever an unvisited tile is queried, the tile may be set to "incomplete," "free," or "blocked." If the queried point is located within a blockage, then the tile is set to "blocked." If no blockage contains the queried point when the query operation reaches one leaf node from the root, PMT extraction and PMT merging are repeated to yield a PMT. All space tiles used in forming the newly extracted PMT are set to "free," and the space tiles that comprise the first extracted PMT with different borders are set to "incomplete" to indicate that the new PMT can potentially merge with its neighboring space tiles to extend its region when it is queried again in routing. As few PMTs as possible can then be extracted.

C. PMT Propagation

The process of PMT propagation is the same as that in tile-based routing. During propagation, a tile legality query

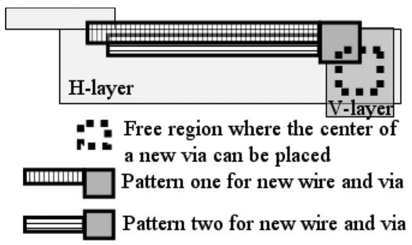
H-layer
V-layer
Free region where the center of a new via can be placed
Pattern one for new wire and via
Pattern two for new wire and via

Fig. 14. PMT propagation does not determine the positions of the wires and vias of a routing path. A heuristic rule can be subsequently applied to optimize resource utilization and then to increase completion rate.

must always precede the next move. If space PMTs abut the current PMT, then path propagation on the same routing layer is conducted. Path propagation to the routing plane of the adjacent layer first involves the *PA* to identify all reachable tiles on the adjacent layer from the current PMT and then performs PMT extraction and merging to produce reachable PMTs on adjacent layer. An overlap check to ensure that the overlapping region can accommodate a new via is performed on the current PMT and the newly generated PMTs.

Theorem 1: The new implicit-connection-graph-based point-to-point routing can find an optimal solution.

Proof: Since routing planes are constructed using the nonzero-width wire model—the same as that used by tile-based routing—the expanded obstacles on routing planes are the same as those on routing planes of tile-based routing. If the routing planes of the new implicit connection graph are considered as being composed of the expanded obstacles and PMTs, then these obstacles and PMTs are exactly equivalent to the routing planes in tile-based routing. By applying piecewise linear cost propagation in [7], the PMT propagation on the new implicit connection graph can also generate an optimal solution.

Since path propagation identifies only a series of connected PMTs (that abut or overlap neighboring PMTs), the determination of the exact positions of the wires and the vias in a routing path is postponed and finished in the path-construction stage. The delay of the decision offers the flexibility of applying simple heuristic rules to modulate the position or pattern of the processed routing path for the purpose of crosstalk reduction or the increase in resource utilization. For example, in the construction of a sensitive wire within a long PMT, breaking a straight wire into a staircase-shaped wire is preferable to a straight wire, because it reduces the length of the wire that is parallel to its neighboring wires. Fig. 14 displays an example of increased resource utilization by the shifting of wires during path construction. Path construction identifies a routing path along three connected PMTs (two H-layer PMTs and one V-layer PMT). The bold dotted rectangle is the legal region in which the centers of all of the potential vias can be placed. Two wiring patterns are available, as shown in Fig. 14. The first aligns the top borders of a wire and a via, and the other aligns the centers of a wire and a via. Based on a search for the centerline path and the center via, the second is straightforward but wastes space. If the first type is employed, the new wire is held close to its top neighboring blockage according to the minimum-space rule. Sometimes, such little space relaxation can complete a hard routing in a congested region during rip-up

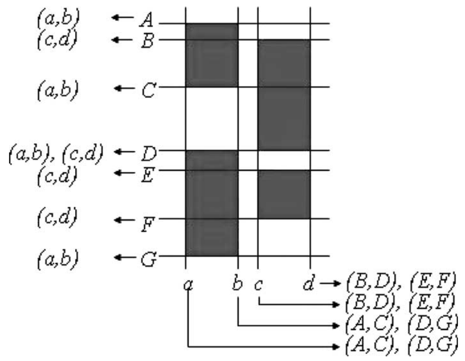

Fig. 15. Each grid line stores the endpoints of all borders by which the gridline is induced.

and rerouting. The adjustment is preferably made in the path-construction stage, rather than in the path-propagation stage.

V. Full-Chip Routing

A. Gridline Reduction and Pseudoblockage Insertion

The detailed routing adheres to the global path. The boundaries of all blockages that fall entirely outside the global path potentially increase the number of gridlines inside the global path. The blockages that overlap or are within the global path are called *active blockages*, whereas the others are called *idle blockages*. Fig. 5(a) illustrates vertical-layer routing paths with a global path whose GCells are filled with a dotted pattern. The figure also shows nine existing blockages. Fig. 5(b) plots the simplified implicit connection graph, whose gridlines are only derived from three active blockages. If the other six idle blockages also induced gridlines, then the routing graph would be very complicated. Furthermore, since the global path boundary is surrounded by pseudoblockages, adjacent PMTs induced by the gridline derived from the idle blockage are likely to have the same height or width such that they can be merged; that is, the result of PMT extraction will not be influenced by the gridlines derived from the idle blockages. Consequently, the gridlines from the idle blockages are redundant, and the result of PMT extraction without gridline reduction is the same as that with gridline reduction. Fig. 5(d) displays the results of PMT extraction with and without considering idle blockages within the global path.

Gridline reduction can be easily implemented. Each gridline is induced by at least one border. When a connection graph is constructed, each gridline records the endpoints of all borders to induce the gridline. Given a global path, gridline reduction involves scanning all gridlines inside the global path. For each gridline, if the attached borders overlap the global path, this gridline is reserved; otherwise, the gridline does not influence the connection graph and is discarded. Fig. 15 presents an illustrative example of the endpoint list. Gridline a is induced by two borders (A, C) and (D, G), and therefore, a stores the endpoints of these two borders.

NEMO adopts pseudoblockages to speed up pseudo tile extraction and constrain the detailed routing. Since tile propagation does not exceed the GCells in the global path, the production of PMTs with borders outside the global path is irrelevant. The extraction of a border outside the global path requires the visiting of additional leaf nodes along the interval tree and is time consuming. Hence, pseudoblockages can reduce the number of visited nodes during PMT extraction, produce PMTs all within the global path, and then forbid path propagation from leaving the global path.

TABLE II
Circuit Statistics for Two-Terminal Single-Net Routing

Test Case	Rectangles		Chip Dimension
	Met2	Met3	x × y (um)
D1	5,106	3,720	470.00 × 455.00
D2	1,650,704	1,018,211	7320.00 × 7320.00

B. Variable-Rule Routing and DSM Design Rules

NEMO processes variable-rule routing by first partitioning and grouping the nets with the same rule into a set, and then performing detailed routing set by set. For each set, NEMO follows the routing flow of the initial routing, which is followed by rip-up and rerouting, to complete the routing of all of the nets in the set. The routing of the next rule set is initiated after the routing of a current set has been completed. Currently, the rip-up strategy considers only the ripping up of the nets, which blocks currently processed nets, to clean the region closest to the target from the present routing result. The nets to be ripped up do not have to be governed by the same design rule as that of the current net. However, if more than one net block the currently processed net, the net that is governed by the same design rule as that of the current net will be chosen.

VI. Experimental Results

A. Single-Net Routing Result

A point-to-point detailed router for two-terminal single-net routing based on the new implicit connection graph with PMT propagation was implemented with the C++ programming language. Two-terminal single-net routings were performed using only the detailed router on a 3.4-GHz Pentium IV PC with 1.7 Gb of random access memory, and two real and different VLSI designs. Table II lists the statistics for two circuits, including the number of rectangles of metal-2 and metal-3 layers and chip dimension.

To compare the point-to-point routing of NEMO with other connection-graph-based gridless routers, the point-to-point gridless detailed router in [11] was implemented with the same basic slit and interval tree data structure as NEMO, but without the enhanced PMT extraction method. Comparison results are obtained by running two routers on the aforementioned platform (Tables III and IV). The point-to-point routings performed on the two cases include short-distance routing, long-distance routing, and hard-to-route routing (only $D2$), in which pins are located in very congested region or which requires numerous layer switches and detours. In Tables III and IV, the second column lists the routing results of the router in [11], whereas the third column lists NEMO results, and the fourth column shows the runtime speedup achieved by

TABLE III
TWO-TERMINAL SINGLE-NET ROUTING RESULTS FOR CIRCUIT $D1$

	Cong[10]			NEMO			
	WL	#Via	RT(Tb)	WL	#Via	RT(Ta)	SU
TEST1	539.50	10	0.719	539.50	10	0.109	6.6
TEST2	382.20	6	0.515	382.20	6	0.047	10.96
TEST3	419.50	12	0.532	419.50	12	0.063	8.44
TEST4	527.50	8	0.734	527.50	8	0.078	9.41
TEST5	649.00	8	0.922	649.00	8	0.094	9.81
Average Speedup							9.04

✧ WL: Wire Length (um), RT: Running Time (second)
✧ SU: Speedup (Tb/ Ta)

TABLE IV
TWO-TERMINAL SINGLE-NET ROUTING RESULTS FOR CIRCUIT $D2$

	Cong[10]			NEMO			
	WL	#Via	RT(Tb)	WL	#Via	RT(Ta)	SU
TEST1	4997	238	73	4997	238	5	14.6
TEST2	9478	366	170	9478	366	38	4.47
TEST3	8299	246	96	8299	246	6	16
TEST4	11794	446	127	11794	446	11	11.55
TEST5	12391	458	166	12375	490	25	6.64
TEST6	10212	374	109	10212	374	8	13.63
TEST7	11538	410	134	11538	410	11	12.18
TEST8	15413	180	329	15413	180	72	4.57
Average Speedup							10.45

NEMO when compared with that of the implicit-connection-graph-based router in [11]. The routing qualities of two routers for cases $D1$ and $D2$ are the same; however, NEMO obtains average speedups of 9.04 and 10.45 for D1 and D2, respectively.

B. Full-Chip Routing

Full-chip routings are performed on a 1.2-GHz Sun Blade-2000 workstation with 2 GB of memory. Table V lists the statistics for eight circuits; the first six designs are benchmark circuits, and the last two designs are of different types. In the table, "#Lay" shows the number of available routing layers, "# 2-pin nets" indicates the number of two-pin connections after net decomposition, and "#GC" shows the number of rows and columns after the chip is partitioned into GCells in the global routing stage.

Table VI displays the routing comparison results for four gridless detailed routers, a three-level routing system featuring a performance-driven global router [22], a noise-constrained wire spacing and track assignment algorithm [23], and, finally, a gridless detailed routing algorithm with wire planning [12], an enhanced multilevel routing system—MARS [14]—a two-pass bottom-up router [19], and NEMO. The gridless detailed routers of the first three routing systems are the implicit-connection-graph-based gridless router. In Table VI, "Comp." denotes the average improvement of NEMO for the first two

TABLE V
BENCHMARK CIRCUITS STATISTICS FOR FULL-CHIP ROUTING

Circuit	Size (μm)	# Lay	# 2-pin nets	#Pin	#GC
S5378	4350 x 2390	3	3124	4818	55 x 30
S9234	4040 x 2250	3	2774	4260	51 x 28
S13207	6600 x 3650	3	6995	10776	83 x 46
S15850	7050 x 3890	3	8321	12793	89 x 49
S38417	11440 x 6190	3	21035	32344	144 x 78
S38584	12950 x 6720	3	28177	42931	163 x 85
B1	22880x12380	3	85838	129376	289x156
B2	25900x13400	3	115554	171724	327x170

items and the total number of failed nets for the last item. Four routers have performed routing on two different platforms, so the runtimes of [12] and [14] require normalization. If runtime is normalized by the clock rate ratio, such as a ratio of 2 for two clock rates 1.2 GHz and 600 MHz, then the runtimes of [12] and [14] needed to be divided by approximately 2.73. Following normalization, NEMO ran at 55.82, 3.93, and 1.69 times faster than the routers in [12], [14], and [19], respectively. The runtime for NEMO included the runtime required for global routing. NEMO obtained a little longer wire length than that of [19]. The spanning-tree-based point-to-point routing in NEMO connects to the end pin and any partially routed wires. The final routing path of a net may be a spanning tree or a Steiner tree. NEMO does not refine the routing result by transforming a spanning tree into a Steiner tree or restructuring a poor-quality Steiner tree. The wire length can be further reduced by refining those spanning-tree-based or poor-Steiner-tree-based routing paths. Table VII compares the routing results of a commercial graph-based router (GBR) and NEMO. All optimization features of GBR, such as timing, crosstalk, and antenna, are disabled. The routing time of GBR increases gradually as the number of nets exceeds 100 000. NEMO gains reduced vantage in routing speed while the circuit size increases and spends 43% more runtime than GBR for the largest case B2. For comparing routing quality, GBR equally uses the routing resources of every routing layer, whereas NEMO simply minimizes routing cost; consequently, NEMO produces fewer vias. On the other hand, GBR is speculated to be an area router and employs a window-based rip-up and rerouting process to solve routing violations. Local rip-up and rerouting has the superiority of rapidly removing violations but may cause detours and vias. NEMO simply rips up and reroutes an entire existing net. This is a simple rip-up and rerouting scheme and may produce less detours and vias but at the cost of increased runtime. Thus, as the circuit size increases, NEMO has higher runtime rising rate than GBR. Substituting complex, yet efficient window-based rip-up and rerouting for the present simple scheme can improve the performance of NEMO by an increasing factor in proportion to the size of the circuit, overcoming the aforementioned shortcoming.

For variable-rule routing, the rules for designing the six benchmark circuits are adapted as follows. The width of the longest 10% of nets is doubled, that of the next 10% of nets is multiplied by 1.5, and the others remain unchanged. Since many pins on the first metal layer are aligned and separated in

TABLE VI
COMPARISON OF ROUTING RESULTS AMONG FOUR GRIDLESS ROUTERS. (a) THREE-LEVEL ROUTING SYSTEM [12]. (b) ENHANCED MULTILEVEL ROUTING SYSTEM [14]. (c) TWO-PASS BOTTOM-UP GRIDLESS ROUTER [19]. (d) NEMO

Circuit	(A)Three-Level Routing [12, 22, 23]			(B)Multilevel Routing [14]		
	Time (s) /T_n	WL (μm)	# of Failed Nets	Time (s) /T_n	WL (μm)	# of Failed Nets
S5378	430/157.7	-	517	30/11	-	0
S9234	355/130.2	-	307	22.8/8.4	-	0
S13207	1099/403	-	877	85.2/31.2	-	0
S15850	1469/538.7	-	978	107.1/39.3	-	0
S38417	3560/1305.5	-	1945	250.9/92	-	0
S38584	7086/2598.5	-	2535	466.1/170.9	-	0
Comp.	**55.82**		7159	**3.93**		0

Circuit	(C) Two-Pass Bottom-Up Routing [19]			(D) NEMO		
	Time(s)	WL (μm)	# of Failed Nets	Time(s)	WL (μm)	# of Failed Nets
S5378	4.0	7.4e4	0	2.4	7.4e4	0
S9234	2.9	5.4e4	0	1.7	5.5e4	0
S13207	11.6	1.8e5	0	6.6	1.7e5	0
S15850	16.4	2.2e5	0	8.8	2.2e5	0
S38417	43.7	4.7e5	0	37.2	4.8e5	0
S38584	148.5	6.6e5	0	73.7	6.7e5	0
Comp.	**1.69**	0.989	0	1	1	1

Note: (A) and (B) were run on a 440-MHz Sun Ultra-10 with 384 MB memory; (C) was run on a 1.2GHz Sun Blade 2000 with 8 GB memory; (D) was run on a 1.2 GHz Sun Blade 2000 with 2 GB memory; Tn: the normalized runtime; WL: Wire length.

TABLE VII
COMPARISON OF ROUTING RESULTS BETWEEN A COMMERCIAL GBR AND NEMO

Circuit	Graph-based router				NEMO			
	Time (s)	WL (μm)	# of Vias	Memory (M)	Time (s)	WL (μm)	# of Vias	Memory (M)
S5378	8	7.5e4	6996	19	2.4	7.4e4	6840	10
S9234	7	5.6e4	5997	19	1.7	5.5e4	5837	9
S13207	21	1.8e5	16035	24	6.6	1.7e5	14731	15
S15850	24	2.2e5	19129	24	8.8	2.2e5	17884	18
S38417	62	4.8e5	45024	35	37.2	4.8e5	48847	48
S38584	88	6.8e5	59556	44	73.7	6.7e5	59402	66
B1	308	3.3e6	193067	86	244.9	3.2e6	182560	124
B2	493	5.6e6	288217	120	703.3	5.3e6	254424	247

TABLE VIII
COMPARISON OF ROUTING RESULTS FROM UNIFORM DESIGN RULES AND VARIABLE DESIGN RULES OBTAINED USING NEMO

Circuit	Uniform design rules				Variable design rules			
	Time(s):T_u	WL (μm)	# of Vias	# of Failed Nets	Time(s):T_v / (T_v/T_u)	WL (μm)	# of Vias	# of Failed Nets
S5378	2.4	7.4e4	6840	0	3.74/1.56	7.6e4	6649	0
S9234	1.7	5.5e4	5837	0	2.69/1.58	5.6e4	5677	0
S13207	6.6	1.7e5	14731	0	11.28/1.71	1.8e5	14593	0
S15850	8.8	2.2e5	17884	0	16.33/1.86	2.2e5	17372	0
S38417	37.2	4.8e5	48847	0	56.36/1.52	4.9e5	47663	0
S38584	73.7	6.7e5	59402	0	143.39/1.94	6.8e5	57922	0

minimum-rule space, and enlarging the rule of the first metal layer and its pins would violate the design rules, the rules of the first metal layer remain unchanged. Table VIII compares the routing results obtained using the uniform design rules and the variable design rules. Variable-rule routing has less than double (from 1.52 to 1.94 times) the runtime of uniform-rule routing. Since the rules for enlarging the design rules yield the same number of rule sets in all test cases, the numbers of reconstruction of routing graphs required in all test cases are the same. The numbers of rip-ups and reroutings for fixed-rule routing and variable-rule routing are similar. Accordingly, the factors by which the runtime of variable-rule routing are increased in all test cases are similar, ranging from 1.52 to 1.94. Variable-rule routing always produced longer wires than uniform-rule

TABLE IX
COMPARISON OF ROUTING RESULTS BETWEEN A MULTILEVEL GRID-BASED ROUTING SYSTEM [18] AND NEMO

Circuit	Multilevel router without antenna avoidance [18]				NEMO			
	Time (s) /T_n	WL (μm)	# of Vias	# of Failed Nets	Time (s)	WL (μm)	# of Vias	# of Failed Nets
S5378	35/29.2	8.2e4	7163	0	2.4	7.4e4	6840	0
S9234	26/21.7	6.0e4	6287	0	1.7	5.5e4	5837	0
S13207	106/88.3	2.2e5	14938	0	6.6	1.7e5	14731	0
S15850	538/448.3	2.4e5	17334	0	8.8	2.2e5	17884	0
S38417	899/749.2	5.9e5	43551	0	37.2	4.8e5	48847	0
S38584	1953/1627.5	7.7e5	61053	0	73.7	6.7e5	59402	0
Comp.	21.91	1.16	1.00	0	1	1	1	0

Note: multilevel system [18] was run on a 1 GHz Sun Blade 2000 with 1 GB memory; NEMO was run on a 1.2 GHz Sun Blade 2000 with 2 GB memory;

routing, but using fewer vias. If the wires that intend to pass through a GCell contain different wire widths, then the order of routed nets from a global router influences the number of nets that utilize this GCell. Because the global router does not consider how to maximize routing-resource utilization, each GCell might contain less congestion than uniform-rule routing, possibly producing fewer vias and longer wire lengths for nets that are forced to generate detoured global paths. Therefore, variable-rule routing produced longer wires with fewer vias than uniform-rule routing in all cases.

Table IX compares NEMO routing with a multilevel grid-based router with antenna avoidance. The data used here are for routing without antenna avoidance. NEMO has superior routing results to that described in [18]—21.91× improvement in routing speed and a 16% reduction rate in wire length.

VII. Discussions

Routing with DSM design rules is an important issue but not currently addressed by NEMO. The impact of DSM technology on the routing problem engenders new routing problems, such as optical proximity correction (OPC)-aware routing, and also complicates conventional design rules. NEMO might be burdened by the DSM rules since some tests on path realization might be required in the early stage of tile propagation instead of in-path construction. The spacing between a wire and its vicinal wires varies according to the wire's maximum parallel length to all neighboring wires. Solving such a complex spacing rule requires path verification during path propagation instead of path construction. The largest spacing rule for contour insertion lowers the available routing resource, but increases the speed of path propagation and correct result discovery. However, it is infeasible due to the low routing completion rate. If the default spacing rule for contour insertion is employed, and tests are then performed on the current tile using the associated large spacing rule based on the possible run length (so that abutting tiles or via regions can be reached) inside the tile, the complex spacing rules can thus be addressed. This approach undoubtedly reduces the routing speed. The minimum-jog rule and the notch rule also require tests to be performed on the path during path propagation. The minimum-jog rule prevents the production of two perpendicularly consecutive wire borders with lengths below a certain value, such as half the wire width. The undesirable jog pattern disturbs the corner correction of the OPC in the postrouting stage. The test on undesirable jogs during tile propagation can prevent further unnecessary propagation. Notch filling is conventionally a postprocessing procedure for removing an unforeseen notch but might cause another fat-wire-space error. For instance, filling a notch generated by two close vias might form a fat wire and then induce an unexpected spacing error. This problem can be solved by avoiding the occurrence of notches as far as possible or dynamically testing notch filling and a fat-wire-spacing check on the current tile and its neighboring space tiles. If these issues are disregarded during tile propagation, then the identified tile list is likely to be useless.

VIII. Conclusion

This paper proposed a new implicit-connection-graph-based gridless detailed router, called NEMO, with a multilayer implicit connection graph, a nonzero-width wire model, and PMT extraction and propagation. For point-to-point routing, NEMO offers the advantages of simple and fast routing-plane construction, efficient PMT extraction, and faster PMT propagation than a grid maze. Experimental results on point-to-point routing indicate that NEMO runs almost ten times faster than the one described in [11], while an optimal path is guaranteed. For full-chip routing, NEMO employs pseudoblockage insertion and gridline reduction to produce routing planes more effectively than before. Integrated with a congestion-driven global router, NEMO improves routing performance by a factor of around 1.69×–55.82× over that obtained in previously published works, based on a set of commonly used MCNC benchmark circuits. For a test case with over a hundred thousands nets, NEMO performs more slowly than a commercial GBR since NEMO presently employs a simple rip-up and rerouting scheme. For variable-rule routing, NEMO obtains longer wires and fewer vias than does the imposition of uniform rules since the global router does not maximize the routing-resource utilization. NEMO also outperforms a multilevel grid-based routing system in routing speed and wirelength.

Acknowledgment

The authors would like to thank the anonymous reviewers, whose detailed and valuable suggestions helped the authors shape and clarify the paper.

REFERENCES

[1] M. Sato, J. Sakanaka, and T. Ohtsuki, "A fast line-search method based on a tile plane," in *Proc. IEEE Int. Symp. Circuits and Syst.*, May 1987, pp. 588–591.

[2] A. Margarino, A. Romano, A. De Gloria, F. Curatelli, and P. Antognetti, "A tile-expansion router," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. CAD-6, no. 4, pp. 507–517, Jul. 1987.

[3] C.-C. Tsai, S. Chen, and W. Feng, "An H–V alternating router," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 11, no. 8, pp. 976–991, Aug. 1992.

[4] J. Dion and L. M. Monier, "Contour: A tile-based gridless router," Western Res. Lab., Palo Alto, CA, Western Research Laboratory Research Rep. 95/3.

[5] L.-C. Liu, H.-P. Tseng, and C. Sechen, "Chip-level area routing," in *Proc. Int. Symp. Phys. Des.*, Apr. 1998, pp. 197–204.

[6] H.-P. Tseng and C. Sechen, "A gridless multilayer router for standard cell circuits using CTM cells," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 18, no. 10, pp. 1462–1479, Oct. 1999.

[7] Z. Xing and R. Kaog, "Shortest path search using tiles and piecewise linear cost propagation," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 21, no. 2, pp. 145–158, Feb. 2002.

[8] A. Hetzel, "A sequential detailed router for huge grid graphs," in *Proc. IEEE Des. Autom. Test Eur.*, Feb. 1998, pp. 332–338.

[9] S. Q. Zheng, J. S. Lim, and S. Iyengar, "Finding obstacle-avoiding shortest paths using implicit connection graphs," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 15, no. 1, pp. 103–110, Jan. 1996.

[10] J. Cong, M. Xie, and Y. Zhang, "An enhanced multilevel routing system," in *Proc. IEEE Int. Conf. Comput.-Aided Des.*, Nov. 2002, pp. 51–58.

[11] J. Cong, J. Fang, and K. Khoo, "An implicit connection graph maze routing algorithm for ECO routing," in *Proc. Int. Conf. Comput.-Aided Des.*, Nov. 1999, pp. 163–167.

[12] ——, "DUNE: A multilayer gridless routing system," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 20, no. 5, pp. 633–646, May 2001.

[13] J. Cong, J. Fang, and Y. Zhang, "Multilevel approach to full-chip gridless routing," in *Proc. IEEE Int. Conf. Comput.-Aided Des.*, San Jose, CA, Nov. 2001, pp. 396–403.

[14] J. Cong, J. Fang, M. Xie, and Y. Zhang, "MARS—A multilevel full-chip gridless routing system," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 24, no. 3, pp. 382–394, Mar. 2005.

[15] Y.-W. Chang and S.-P. Lin, "MR: A new framework for multilevel full-chip routing," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 23, no. 5, pp. 793–800, May 2004.

[16] T.-Y. Ho, Y.-W. Chang, S.-J. Chen, and D. T. Lee, "Crosstalk- and performance-driven multilevel full-chip routing," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 24, no. 6, pp. 869–878, Jun. 2005.

[17] T.-C. Chen and Y.-W. Chang, "Multilevel full-chip gridless routing considering optical proximity correction," in *Proc. ACM/IEEE ASP-DAC*, Shanghai, China, Jan. 2005, pp. 1160–1163.

[18] T.-Y. Ho, Y.-W. Chang, and S.-J. Chen, "Multilevel routing with antenna avoidance," in *Proc. Int. Symp. Phys. Des.*, Apr. 2004, pp. 34–40.

[19] H.-Y. Chen, M.-F. Chiang, Y.-W. Chang, L. Chen, and B. Han, "Novel full-chip gridless routing considering double-via insertion," in *Proc. 43rd Des. Autom. Conf.*, Jul. 2006, pp. 755–760.

[20] J.-Y. Li and Y.-L. Li, "An efficient tile-based ECO router with routing graph reduction and enhanced global routing flow," in *Proc. ACM ISPD*, San Francisco, CA, Apr. 2005, pp. 7–13.

[21] J. Ousterhout, "Corner stitching: A data-structuring technique for VLSI layout tools," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. CAD-3, no. 1, pp. 87–100, Jan. 1984.

[22] J. Cong and P. Madden, "Performance driven multilayer general area routing for PCB/MCM designs," in *Proc. 35th Des. Autom. Conf.*, Jun. 1998, pp. 356–361.

[23] C. Chang and J. Cong, "Pseudo pin assignment with crosstalk noise control," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 20, no. 5, pp. 598–611, May 2001.

Yih-Lang Li (M'04) received the B.S. degree in nuclear engineering and the M.S. and the Ph.D. degrees in computer science from the National Tsing-Hua University, Hsinchu, Taiwan, R.O.C., in 1987, 1990, and 1996, respectively.

In February 2003, he joined the faculty of the Department of Computer Science, National Chiao-Tung University (NCTU), where he is currently an Assistant Professor. Prior to joining the faculty of NCTU, from 1995 to 1996 and from 1998 to 2003, he was a Software Engineer and an Associate Manager at Springsoft Corporation, Hsinchu, where he was heavily involved in the development of verification and synthesis tools for custom-based layout. His research interests include physical synthesis, parallel architecture, and VLSI testing.

Hsin-Yu Chen received the B.S. degree in computer science and information engineering from Tunghai University, Taichung, Taiwan, R.O.C., in 2002, and the M.S. degree in computer and information science from the National Chiao-Tung University, Hsinchu, Taiwan, R.O.C., in 2005.

He is currently with Faraday Technology Corporation, Hsinchu.

Chih-Ta Lin received the B.S. degree in computer and information science from the National Chiao-Tung University, Hsinchu, Taiwan, R.O.C., in 2004. He is currently working toward the Ph.D. degree at the Department of Computer Science, National Chiao-Tung University.

His research interests include physical design automation.