MARS—A Multilevel Full-Chip Gridless Routing System

Jason Cong, *Fellow, IEEE*, Jie Fang, Min Xie, *Student Member, IEEE*, and Yan Zhang, *Student Member, IEEE*

Abstract—This paper presents MARS, a novel multilevel full-chip gridless routing system. The multilevel framework with recursive coarsening and refinement allows for scaling of our gridless routing system to very large designs. The downward pass of recursive coarsening builds the representations of routing regions at different levels while the upward pass of iterative refinement allows a gradually improved solution. We introduced a number of efficient techniques in the multilevel routing scheme, including resource reservation, graph-based Steiner tree heuristic and history-based iterative refinement. We compared our multilevel framework with a recently published three-level routing flow [1]. Experimental results show that MARS helps to improve the completion rate by over 10%, and the runtime by $11.7\times$.

Index Terms—Design automation, routing optimization methods, very large scale integration (VLSI).

Fig. 1. Traditional two-level routing flow.

I. INTRODUCTION

THE continuous increase of the problem size of IC routing has become a great challenge to existing routing algorithms. The traditional method for handling the large problem size is to "divide-and-conquer," which breaks the routing problem into two successive steps, global routing and detailed routing, as shown in Fig. 1.

Global routing partitions the entire routing region into tiles or channels, and tries to find a tile-to-tile path for each net with congestion and performance optimization. There are two kinds of global routing algorithms. Sequential methods route the nets one-by-one in a predetermined order, using either the maze searching algorithm [2], [3] or the line-probe algorithm [4], [5]. However, the solution quality is often affected by the net ordering. Iterative methods try to overcome the net ordering problem by performing multiple iterations. The negotiation-based iterative global routing scheme was proposed in [6], and later on used in field programmable gate array (FPGA) routing [7]. The multicommodity flow-based iterative methods were also proposed [8], [9], where the global routing problem is modeled as a multiterminal, multicommodity flow problem and approximate solutions are computed iteratively. A more recent work used a combination of maze searching and iterative
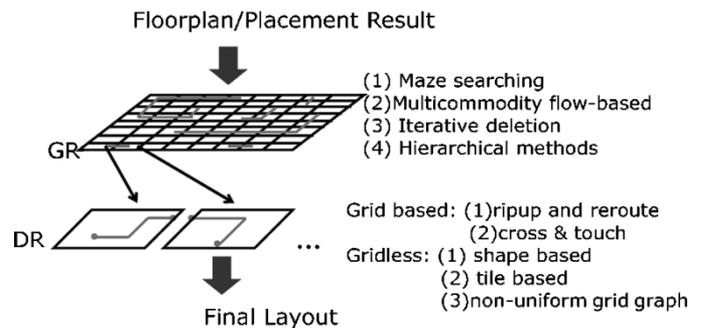
deletion for solving the performance-driven global routing problem [10].

After global routing is completed, detailed routing is performed within each tile or channel, where the exact geometric layout of all nets is determined. There are mainly two types of detailed routing approaches, grid-based and gridless routing algorithms. In grid-based detailed routing, routing grids are defined before the actual routing process, with fixed spacings according to the design rule. The path of each net is confined to the grids. Since the grids are uniform, the layout representation in grid-based routing is quite simple, usually in the form of a three-dimensional (3-D) array. Variable widths and variable spacings may be used for delay and noise minimization (e.g., see [11], [12]). A gridless detailed router allows arbitrary widths and spacings for different nets, which can help to optimize the circuit performance and to reduce noise. However, the design size that a gridless router can handle is usually limited, due to the high complexity of the routing problem.

Most global routing algorithms run directly on a two-dimensional (2-D) or 2.5-dimensional array of routing tiles, such flat two-level routing approaches (global routing + detailed routing) have two limitations in current and future very large scale integration (VLSI) designs. First, future designs may integrate over several hundreds of millions of transistors in a single chip. Traditional two-level design flow may not scale well to handle problems of such size. For example, a 2.5×2.5-cm^2 chip in the 0.07-μm processing technology, as predicted by [13], may have over $360\,000$ horizontal and vertical routing tracks at the full chip level. That will translate to about 600×600 routing tiles if we balance the problem size between the global routing and detailed routing stages, which presents a significant challenge to the efficiencies of both stages. To handle the problem, several routing approaches have been proposed to scale to large circuit designs. We shall summarize them briefly as follows.
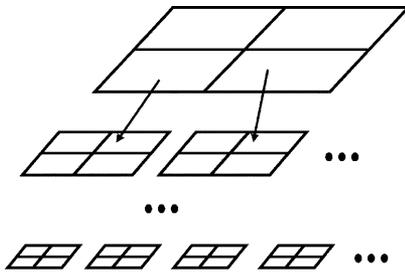
Fig. 2.   Hierarchical routing flow.

Fig. 3.   Three-level routing flow.

Approaches with multiple levels of hierarchy have been used to handle large routing problems, whose flow is shown in Fig. 2. The first top-down hierarchical method was proposed for channel routing by Burstein [14]. Heisterman and Lengauer proposed a hierarchical integer programming-based algorithm for global routing [15]. Wang and Kuh proposed a hierarchical $(\alpha, \beta)^*$ [16] algorithm for the MCM global routing. Instead of the top-down approach, Marek-Sadowska proposed a bottom-up hierarchical routing method [17]. The problems with both the top-down and the bottom-up hierarchical approaches are: 1) the previous level solutions will constrain the later level solutions and 2) the lack of the routing information of the future levels available at the current levels makes it difficult to make well-informed decisions. When an unwise decision is made at some point, it is very costly (through ripup-and-reroute) to revise it at a later stage. To overcome the disadvantages of hierarchical methods, hybrid routing systems are proposed. Lin *et al.* proposed a combined routing approach of a maze-routing algorithm and a top-down hierarchical method [18]. Parameter-controlled expansion instead of strictly confined expansion is used to overcome the first disadvantage, but there is still no way to learn finer level routing information at coarser levels. Hayashi and Tsukiyama proposed a combination of a top-down and a bottom-up hierarchical approach [19], aimed at resolving the second problem of the original hierarchical approach, while the fine level planning results are still fixed once they are generated, causing the net-ordering problem.

In a recent work [1], a three-level routing scheme with an additional wire-planning phase between the performance-driven global routing and the detailed routing was proposed for deep submicron very large scale integration (VLSI) routing, as shown in Fig. 3. The additional planning phase improved both the completion rate and the runtime. However, for large designs, even with the three-level routing system, the problem size at each level may still be very large. For the previous example of a 2.5×2.5-cm^2 chip in the 0.07-μm processing technology, the routing region has to be partitioned into over 100×100 tiles on both the top-level global routing and the intermediate-level wire planning stage (assuming the final tile for detailed routing is about 30×30 tracks for the efficiency of the gridless router). Therefore, as the designs grow, more levels of routing are needed. Rather than a predetermined, manual partition of levels, which may have discontinuity between levels, an automated flow is needed to enable seamless transitions between levels.

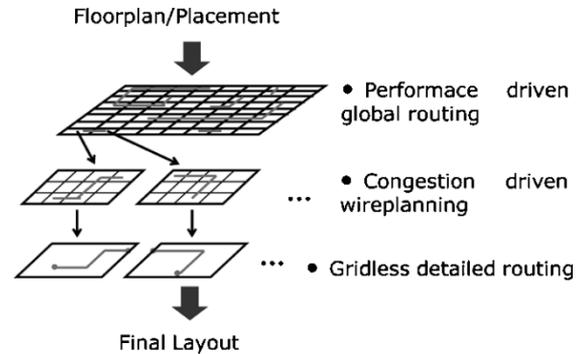Following our multilevel routing idea, another group of researchers subsequently proposed another multilevel routing framework [20], [21], which combines the global routing and the detailed routing together. However, the routing system in [20], [21] is grid-based approach, which is quite different from our detailed routing algorithm in terms of the layout database management, overall framework and the target problem.

In this paper, we propose MARS, an enhanced multilevel routing framework, for the gridless routing problem of large VLSI designs. Experimental results show that compared to our recent three-level routing system [1], MARS helps to improve the completion rate by over 10%, and the runtime by $11.7\times$. The paper is organized as the follows. Section II provides an overview of our multilevel routing framework. Section III explains our tile partitioning and resource estimation and reservation method. A coarsening process generates the level representations from the finest level to the coarsest level. An approximate multicommodity flow algorithm is used to compute the initial routing solution at the coarsest level in Section IV. A history-based iterative refinement scheme is presented in Section V for uncoarsening in the multilevel routing flow. Finally, the effectiveness of our algorithm is validated with experimental results in Section VI. The paper concludes with a discussion of several possible extensions of the proposed multilevel framework for VLSI routing in Section VII. The preliminary versions of this work were reported in [22] and [23].

II. OVERVIEW OF THE MULTILEVEL ROUTING SYSTEM

The multilevel method was originally used as a means of accelerating numerical algorithms for partial differential equations (e.g., [24] and [25]). In this past decade, it has been also applied to other areas, such as image processing, combinatorial optimization, control theory, statistical mechanics, quantum electrodynamics, and linear algebra. Multilevel techniques for VLSI physical designs have recently shown promising results. Good progress has been made in multilevel circuit partitioning and placement. The multilevel partitioning algorithm *hMETIS* [26] produces the best cut size minimization in circuit partitioning. The multilevel performance-driven partitioning algorithm *HPM* [27] produces the best balance of delay and cut size minimization results for circuit partitioning. The multilevel placement algorithm *mPL* [28] achieves comparable circuit placement quality with the well-known *Dragon* package [29] with over $6\times$ speed-up on designs with over 200 K movable
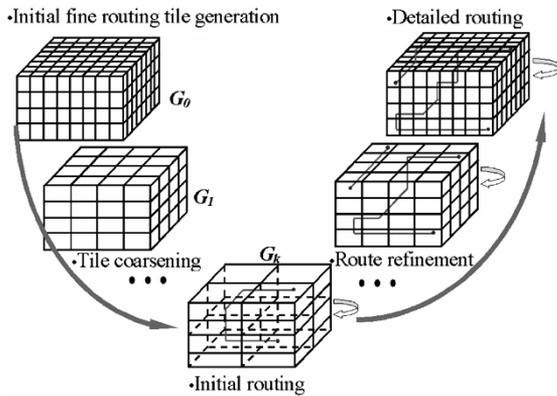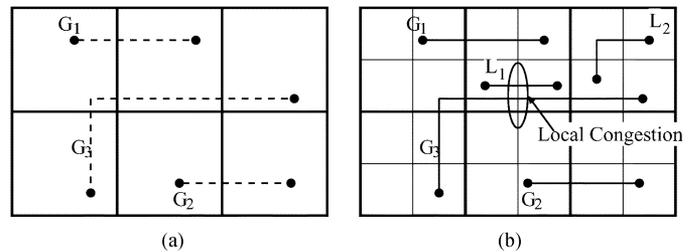
Fig. 4. Multilevel routing flow.

Fig. 5. Limitation of hierarchical approaches. In a hierarchical approach, coarse-level decisions are based on limited information, as shown in (a). However, fine-level refinement lacks the flexibility to change the coarse-level results. Thus, we have a local congestion after the refinement, as shown in (b). (a) Coarse routing result. (b) Refinement at next level.

objects. These successes led us to investigate the application of the multilevel scheme to handling large VLSI routing problems.

MARS features an iterative coarsening algorithm and an iterative refinement algorithm in a "V-shaped" flow (see Fig. 4), which is typical for multilevel optimization. On the downward pass, the design is recursively coarsened, and an estimation of routing resources is calculated at each level. At the coarsest level, a multicommodity flow algorithm is used to generate an initial routing result. On the upward pass, a modified maze searching algorithm is carried out iteratively to refine the results from level to level. The final results of the multilevel planning algorithm are tile-to-tile paths for all the nets. These paths are used to guide the gridless detailed routing algorithm to find the exact connection for each net.

Fig. 4 illustrates a multilevel framework for VLSI routing. The routing area is partitioned into routing tiles. The algorithm goes through a multilevel planning procedure to find a tile-to-tile path for each net among these tiles. In contrast, most traditional global routing algorithms [8]–[10], try to find routing solutions directly on the finest tiles. For large designs, the number of tiles may be too great for these algorithms to handle.

The multilevel approach first accurately estimates the routing resource using a line-sweeping algorithm on the finest tile level. A recursive coarsening process is then employed to merge the tiles, to reserve the routing resource for local nets, and to build coarser and coarser level representations. At each coarsening stage, the resource of each tile is calculated from the previous finer level tiles forming the current tile. Also, part of the routing resource is assigned to nets that are local to that level. This coarsening process is known in multilevel literature as the "downward pass."

Once the coarsening process has reduced the number of tiles to below a certain threshold, the initial routing is computed using a multicommodity flow-based algorithm. A congestion-driven Steiner tree structure is used to gradually decompose multipin nets into two-pin nets. The initial routing result is refined in the reverse direction of coarsening, known as the "upward pass," by a modified maze searching algorithm. The refinement can be repeated at each level when necessary. When the final tile-to-tile paths are found at the finest level of tiles for all the nets, the gridless detailed routing algorithm [30] is applied to find the exact path for each net.

Compared to the flat approaches, the multilevel algorithm is much more scalable to large designs. Traditionally, hierarchical approaches [15], [16] are also used to overcome the scaling problem of large designs. These methods also build multilevel hierarchical representations of the routing region, however, from the coarsest level to the finest level. The key problem with the hierarchical approaches is the lack of detailed routing information available to make routing decisions at the coarse level, while these coarse-level decisions *constrain* the fine-level solutions. Thus, if an unwise decision is made at any level, it is almost impossible, or very costly, to revise it at a finer level in hierarchical approaches, as illustrated in Fig. 5.

The "V-shaped" flow of a multilevel approach is more flexible than a hierarchical approach. The coarsening pass builds up a series of subproblems of different granularities, which is similar to the hierarchical approach, yet the subproblems are closer to the original problem because in each level, all nets, including those local to the current level, are considered by the means of resource reservation. The uncoarsening pass allows the finer level router to *refine* the coarser level result. The coarser level solution only provides a *guide* (as opposed to a *constraint* in the case of hierarchical routing) to the finer level path searching. Therefore, the fine level path searching algorithm has the flexibility to deviate from the coarse level paths when more detailed information about local resource and congestion is considered. These features make the multilevel method converge to better solutions with higher efficiency.

III. DOWNWARD PASS: ROUTING RESOURCE ESTIMATION AND RESERVATION

The first step in our multilevel flow is to build up the multiple levels of routing region representations of coarser and coarser granularity along the "downward pass," which is called the coarsening.

Before the coarsening process starts, the routing region is first partitioned into a 3-D array of fine tiles, each with similar height and width. We denote this level as level 0. Pins are usually aligned to horizontal or vertical lines, experiments show that better results can be achieved if the routing tiles are partitioned so that pins are located in the middle of the tiles rather than at the boundaries. The main reason for this is the detailed routing engine would have more freedom to connect to the pins if they are located at the center of the routing tiles.
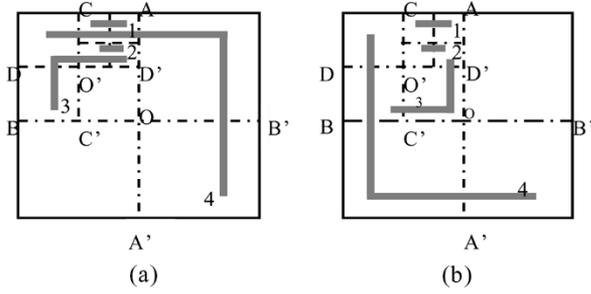
Fig. 6. Resource estimation model.

Fig. 7. Path cost example.

Fig. 8. Merging of contour list.

We then build a three-dimensional routing graph on level 0, denoted as G_0, such that each node in G_0 represents a tile in some routing layer at level 0. Every two neighboring nodes in G_0 that have a direct routing path between them are connected with an edge. The edge capacity represents the routing resource at the common boundary of the two tiles. The ultimate objective of the multilevel routing algorithm is to find a tile-to-tile path for each net in G_0, which is used to guide the gridless detailed router in searching a connection for each net.

The multilevel router first accurately estimates the routing resources at the finest level, then recursively coarsens the representations on different levels.

The same resource estimation model as in [1] is used in the MARS router. All layout objects such as obstacles, pins and pre-routed wires are counted in the calculation. Due to the gridless nature of our routing problem, we use actual dimensions of the obstacles to compute the routing resources. Three kinds of edge capacities are computed, including wiring capacity, interlayer capacity, and through capacity.

To calculate the wiring capacity at the tile boundary, we use a line-sweeping algorithm, similar to the estimation algorithm used in [1], to compute the wiring capacities. The sweeping algorithm cuts the routing region into horizontal (or vertical) empty rectangles called *slices*. For the example, in Fig. 6, the wiring capacity at boundary B can be computed as a *weighted* sum of widths of empty slices along B, by the following formula:

$$C = \sum_i W_i \times D_i/D \qquad (1)$$

where W_i and D_i are the width and depth of each slice S_i, D is the tile depth. To calculate W_i and D_i, we maintain a *contour list* of B, which is defined as a sorted list of the boundaries of all the rectangular obstacles that can be seen from B. In Fig. 6, the *contour list* of B is C_1, C_2, C_3, C_4.

The interlayer edge capacity, which corresponds to the resource that is used by vias, is calculated by the sum of the areas of all empty spaces in the tile. The through capacity, which corresponds to the paths that go straight through a routing tile, is the sum of the boundary capacity contributions of those empty rectangles that span the whole tile. The through capacity at the horizontal direction of the tile in Fig. 6 is $C_{\text{th}} = W_2 + W_4$.

All three capacities contribute to the path costs. For the example in Fig. 7, the total path cost $C_{\text{path}} = c_{1,\text{right}} + c_{2,\text{left}} + c_{2,\text{right}} + c_{2,\text{through}} + c_{3,\text{left}} + c_{3,\text{up}} + c_{4,\text{down}} + c_{4,\text{right}} +$

$c_{5,\text{left}}$, where $c_{1,\text{right}}, c_{2,\text{left}}, c_{2,\text{right}}, c_{3,\text{left}}, c_{4,\text{right}},$ and $c_{5,\text{left}}$ are the costs related to the wiring capacities of tiles 1, 2, 3, 4 and 5, $c_{2,\text{through}}$, is the cost corresponding to the through capacity of tile 2, and $c_{3,\text{up}}, c_{4,\text{down}}$ are the via costs related with the interlayer capacities.

Given the accurate routing capacity estimation at the finest level and the grid graph G_0 that stores such information, the coarsening process generates a series of reduced graph G_i consecutively, each representing a coarsened level i routing problem, P_i, with a different resolution. At a coarser level (level $i + 1$), the tiles are built from the finer level tiles (level i) by merging the neighboring *component tiles*. The coarse level graph, G_{i+1}, which represents the routing resources available at the coarse tiles, can also be derived from the fine level graph G_i directly. We iteratively coarsen the tiles and the routing graphs until the size of the graph falls below a predetermined threshold.

A. Merging Resource

Every move from a finer level i to a coarser level $i + 1$, requires merging a certain number of component tiles (2×2 in our implementation), into a large one. A new contour list of the resulting tile is obtained by merging the contour list of each component tile. Then, the wiring capacities of the new tile can also be derived by (1). Fig. 8 illustrates the merging process. Level i tiles $T_1, T_2, T_3,$ and T_4, whose left boundaries are $B_1, B_2, B_3,$ and B_4, respectively, are to be merged. The contour list of $B_1, B_2, B_3,$ and B_4 are retrieved and merged into the contour list of the new edge B. Since the contour lists are sorted, the merging process can be accomplished in $O(n)$ time, where n is the number of segments in the new contour list. With the contour list of B, it is straightforward to derive the rectangles abutting B and then calculate the wiring capacity of B. The interlayer capacity of the new tile is calculated as the sum of the interlayer capacities of the component tiles. The through capacity of the new tile is calculated as the sum of the heights of the empty slices that span throughout the entire tile.

Fig. 9. Effect of resource reservation. (a) Effect of local nets (without resource reservation). (b) After resource reservation.

Fig. 10. Reservation calculation.

B. Resource Reservation

The estimation computed by the above procedure, however, still can not precisely model the available routing capacities at the coarser granularity, as when the planning engine moves to a coarser level, a subset of the nets in level i might become completely local to one tile, and thus "invisible" at level $i+1$ and coarser. In hierarchical methods, no effort was made to model these nets, relying on the assumption that they are relatively short and negligible. However, if the number of such local nets is large, a solution to the coarse level problem may not be aware of locally congested areas, which leads to poor planning results.

Fig. 9(a) shows an example of the effect of local nets. Nets 1 and 2 are local to level 1, and appear at level 0. Net 3 is local to level 2, and net 4 is global to the coarsest level (level 2). Each net is planned without any information about the nets local to the planning level. Net 3 and net 4 will be planned as in Fig. 9(a). We can note that both net 3 and net 4 have to be changed in level 0 planning to minimize the local congestion, which not only places a heavier burden on the refinements at later levels, but also wastes the effort spent on the coarser levels.

In order to cope with the above problem, we further predict the portion of the resource that would be used by nets that are local to each level, and then reserve the corresponding amount of resource for those nets explicitly. This process is called *resource reservation*.

More specifically, suppose the coarsening process goes through level 0, level 1 , ..., level k, with level 0 being the finest level. Let $c_{i,j}$ denote the initial capacity of edge $e_{i,j}$ in routing graph G_i, and the capacity vector $C_i = [c_{i,1}, c_{i,2}, \ldots c_{i,m}]$ represents all routing capacities at level i. Let $T_{n,i} = \{$the set of tiles to which the pins of net n are located on level $i\}$, which is called the *spanning tile set* of net n on level i. The level of net n, $\text{level}(n)$, is defined as the level above which a net becomes within the boundary of one tile. $\text{level}(n)$ can be calculated as

$$\text{level}(n)$$
$$= \begin{cases} k, & \text{if } |T_{n,k}| > 1 \\ -1, & \text{if } |T_{n,0}| = 1 \\ \min\{i||T_{n,i}| > 1 \quad \text{and} \quad |T_{n,i+1}| = 1\}, & \text{otherwise.} \end{cases}$$
(2)

Let $L_i = \{n|\text{level}(n) = i\}$, L_i is called the *local net set* on level i.

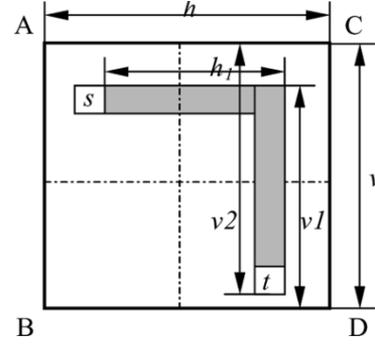Let $M_i = \{n|\text{level}(n) > i\}$, M_i is called the *global net set* on level i.

To better estimate the local nets, we use a maze-routing engine to find a path for each net in L_i, before going from level i to level $i+1$. Then, we deduct the routing capacity taken by these local nets in resource reservation. Fig. 10 shows an example of the calculation of the reservation at edge CD, AC, and BD of a level $i+1$ tile. s and t are pins on a horizontal layer. An L-shaped path connects s and t. The wiring capacities on CD, AC, and BD are first calculated by (1). After the horizonal wire is added, one segment in the contour list of CD will be pushed right by h. Therefore, the reserved capacity $r = w*h_1/h$, where w is the wire width. Similarly, the vertical resource reservations on AC and BD are $w*v_1/v$ and $w*v_2/v$, respectively. However, since pins are treated as obstacles in contour list generation, the capacity reservation on AB remains zero. A vector $R_{i+1} = [r_{i+1,1}, r_{i+1,2}, \ldots r_{i+1,j}]$ can be obtained by repeating this process, each member corresponds to the reservation on $e_{i+1,j}$ in G_{i+1}. The routing capacity of edges in G_{i+1} is updated as $C_{i+1} = C_{i+1} - R_{i+1} = [c_{i+1,1} - r_{i+1,1}, c_{i+1,2} - r_{i+1,2}, \ldots c_{i+1,j} - r_{i+1,j}]$.

Fig. 9(b) shows the planning result with the resource reservation approach. Net 1 and net 2 are routed at level 0, and the reservations are made for them on CO' and AD'. On level 1, net 3 will take a route different from that of Fig. 9(a), since there is resource reservation for local nets on level 0. For the same reason, net 4 is routed on level 2 in the less congested tiles.

Since the spanning tiles of each net are at most two tiles away from one another, the maze routing engine will not explore many nodes before it reaches the destination, so the reservation procedure is very fast.

One possible drawback would be that the local nets are unnecessarily treated with higher priorities. However, the routes taken during this phase are usually short and straight, so the reservation amounts are probably the lower bounds of the resources actually needed by the nets. Furthermore, the reserved routes are not taken as fixed. They can be changed when necessary during the refinement process.

IV. COARSEST LEVEL: MULTICOMMODITY FLOW-BASED INITIAL ROUTING

After the routing tiles are coarsened to a certain level, the coarsening process stops. A set of tile-to-tile paths is computed for the nets crossing the coarsest tile boundaries. This process is called the *initial routing*, which is quite important to the final result of multilevel routing. First, long interconnects that span

more tiles are among the nets that appear during the initial routing. Normally, these long interconnects are timing-critical and may also suffer noise problems due to coupling along the paths. Initial routing algorithms should be capable of handling these performance issues. Second, the initial routing result will be carried all the way down to the finest tiles through the refinement process in the multilevel scheme. Although a multilevel framework allows finer level designs to change coarser level solutions, a bad initial routing solution will slow down the refinement process and may even degrade the final solution.

In MARS, a multicommodity flow-based algorithm is used to compute the initial routing solution at the coarsest level (level k). Several existing routing algorithms use the multicommodity flow-based algorithm (e.g., [8] and [9]). We chose the multicommodity flow-based algorithm rather than a net-by-net approach using a maze-searching algorithm or an iterative-deletion algorithm for several reasons. First, the flow algorithm is fast enough for a relatively big grid size. Although in theory we can continue coarsening the tiles so that the number of tiles is very small, the coarsest level should have reasonable granularity so that the initial routing results can influence the later refinement process. Second, the flow algorithm considers all the nets at the same time. This removes part of the net ordering problem in the net-by-net approaches. A globally good solution for *all* nets is especially important at the coarsest level because its solution will be carried to influence all finer-level solutions through the refinement process. Last, a flow algorithm can be integrated with other optimization algorithms to consider special requirements of certain critical nets. For example, we can compute high-performance tree structures such as A-Tree [31], Steiner tree [32] or buffered tree [33], etc., as the candidate tree structures in the flow-based initial routing.

The objective of the initial routing is to minimize the congestion on the routing graph G_0, which represents the coarsest tiles. We first compute a set of candidate trees for each net on the coarsest level routing graph G_k. For a given net i, let $P_i = \{P_{i,1}, \ldots, P_{i,li}\}$ be the set of possible trees. Our current implementation does not consider delay minimization, and focuses mainly on routability and wirelength optimization. Therefore, we use only the graph-based Steiner tree as the candidates for each net. Assume the capacity of each edge on the routing graph is $c(e)$, and $w_{i,e}$ is the cost for net i to go through edge e. Let $x_{i,j}$ be an integer variable with possible values 1 or 0, indicating if path $P_{i,j}$ is chosen or not ($1 \leq j \leq l_i$). Then, the initial routing problem can be formulated as a mixed integer linear programming problem as follows:

$$\text{subject to } \min \lambda$$
$$\sum_{i,j:e \in P_{i,j}} w_{i,e} x_{i,j} \leq \lambda c(e), \quad \text{for } e \in E$$
$$\sum_{j=1}^{l_i} x_{i,j} = 1, \quad \text{for } i = 1, \ldots, n_k$$
$$x_{i,j} \in \{0,1\}, \quad \text{for } i = 1, \ldots, n_k \qquad (3)$$

where n_k is the number of nets to be routed at level k. Normally, this mixed integer programming problem is relaxed to a

```
parameter initialization;
for each iteration {
    for each net {
        if ((there is no candidate topology for this net) ||
            (the cost of the net's last topology in current
                graph increases too much))
            generate a new topology T for this net;
        else
            keep the last topology T;
        assign a unit flow to T;
        update the routing graph edges;
    }
}
pick one topology for each net by randomized
rounding according to the assigned flow volume;
```

Fig. 11. Approximate multicommodity flow algorithm.

linear programming problem by replacing the last constraint in (3) with:

$$x_{i,j} \geq 0, \quad \text{for } i = 1, \ldots, n_k; \quad j = 1, \ldots, n_k. \qquad (4)$$

The relaxed LP problem is called a fractional global routing problem, which can be solved optimally. However, people are usually more interested in faster approximate methods. The maximum multicommodity flow-approximation algorithm is proved to be able to compute the fraction value of $x_{i,j}$ for each net in the above linear programming formulation [34]. Traditionally, the maximum flow-approximation algorithm picks a path and routes a unit flow along the path. Then, it multiplies the length of every edge on this path by $1 + \epsilon$ with a fixed ϵ. Our implementation of fraction computation (Fig. 11) is faster because it uses the approximation method proposed in [9]. In this algorithm, a maximum s-t flow is computed using a faster and more straightforward method: after picking a path, instead of routing the path with a unit flow, we increase the flow along the path as much as possible to saturate the minimum capacity edge along the path. Garg and Konemann [34] proved the error bound of this method and gave a detailed explanation of its application to multicommodity flow computation.

After the fractional results for each path are computed, we need to map the fractional results to integer results. Our algorithm calls a randomized rounding algorithm to convert the fractional values into 0 or 1 values for candidate paths of each net so that one path is chosen for each net. The randomized rounding approach for global routing was first used in [35] and an error bound was estimated. The rounding process is simply repeated many times, such as 100 times, and the best result will be selected as the initial routing result. Experiments show that rounding 100 times can reduce the resulting congestion by 10% to 50%. Using this simple heuristic, we can quickly get an integral solution. This method does not guarantee that there is no overflow at the tile boundaries. In general, overflow can be corrected by rip up and reroute. Our implementation, however, does not use rip up and reroute at this level, because congestion is not a significant problem, and we rely on subsequent refinement steps to remove the possible overflow.

V. UPWARD PASS: HISTORY-BASED ITERATIVE REFINEMENT

One major difference between the hierarchical routing and multilevel routing approaches is that a multilevel framework allows the finer level to *change* coarser-level routing solutions. In the upward pass of the multilevel framework, paths computed by the initial flow-based algorithm are refined from level to level until the finest tiles are finally reached.

A. Graph-Based Steiner Tree Generation and Refinement

Usually, rectilinear Steiner trees (RSTs) are used in the decomposition of multipin nets in the global routing phase. Since the problem of minimum RST has been proved to be NP-hard, many heuristic algorithms such as in [32] and [36] were proposed to get approximate solutions. Most of the Steiner tree approximation algorithms are geometric distance based.

Congestion-driven planning requires the generation of a graph-based Steiner tree structure, which considers congestion as well as wirelength. It is also necessary that the global routing engine be aware of large hard obstacles as well as the congestion caused by wires. Graph-based Steiner tree generation is more time-consuming than a simple Manhattan distance-based tree. The preprocessing procedure of computing all-pair shortest paths to assign graph edge weights is of $O(n^3)$ time complexity.

In [37] and [38], Steiner tree approximation algorithms that consider congestion are proposed, yet the topologies generated are limited to the initial geometric-based spanning tree structure and may not work well for examples with large obstacles. A graph-based A-tree algorithm is proposed in [31], which can avoid large obstacles. The runtime is also reasonable, since during the construction of an A-tree, all-pair shortest paths searching preprocessing is not necessary. However, an A-tree is limited to optimize the paths from the source to all targets and the total wirelength of an A-tree depends on the position of the source. Therefore, the graph-based A-tree topology may not be suitable for the decomposition of noncritical nets.

In MARS, a congestion-driven Steiner tree is used to decompose a multipin net for better wirelength and routability. An A^* point-to-path maze searching algorithm is used during the initial routing and the refinement phase to help the tree generation. Fig. 12 shows the procedure of constructing an initial Steiner tree during the initial routing at the coarsest level. We first decompose the multipin net by a simple Manhattan-distance minimum spanning tree, then sort the edges (two-pin nets) of the spanning tree by their lengths in nondecreasing order, and route each edge by an A^* searching algorithm. The heuristic here is to let longer edges have better chance to become a Steiner edge. Instead of routing from one point to another, the searching process stops whenever any existing path of the current multipin net connecting to the target point is touched.

During each refinement process, we further continue the Steiner tree construction by the modified point-to-path maze searching algorithm. For spanning tree decomposition method, the exact locations of the two end points of each net are fixed. Therefore, the calculation of the refined pin locations are straightforward after the move from level $i + 1$ to level i. While for Steiner tree decomposition, the Steiner point locations are floating, and are constrained by other edges of the same

```
given a net N, find MST;
sort all MST edges by distance;
for all edges in distance increasing order {
    set source and target to the 2 pins;
    if the source(target) has already been
        connected to other nets, change the
        source(target) to the connected paths;
    use a A* Dijkastra search algorithm to find
        a path from source to target;
    if there are multiple minimum paths, choose
        the one that is closest to the center of all
        pins of the net;
}
```

Fig. 12.   Graph-based Steiner tree generation.

multipin net, which are routed before the current edge in the previous level of routing. To solve this problem, we keep an ordering of all routed edges within each multipin net. During the refinement, the local nets are routed first, and the global nets are refined according to the ordering we get from the previous level.

Fig. 13 shows the formation of a Steiner tree of a 5-pin net from level 2, where the multicommodity flow-based initial routing takes place, at level 0, with one big obstacle shown in dark area. The label beside each edge shows the ordering of that edge. At level 2, two edges, $\overline{ab}$ and $\overline{cb}$, are routed. At level 1, two new pins, e and d, and two new edges $\overline{ae}$ and $\overline{bd}$ appear. $\overline{ae}$ and $\overline{bd}$ are routed first, and inserted into an edge (two-pin net) list L. Then, the nets $\overline{ab}$ and $\overline{cb}$ are refined according to the order generated at level 2 and then inserted into L as well. Since one tile is blocked by the big obstacle, the route of net $\overline{cd}$ takes a detour. The resulting L at level 1 is $(\overline{ae}, \overline{bd}, \overline{ab}, \overline{cd})$. At level 0, no new 2-pin nets appear, so all the existing edges are refined according to L from level 1. When the maze router searches paths for nets $\overline{ab}$ and $\overline{cd}$, path of net $\overline{bd}$ is touched before the target points. So, two Steiner points T' and T' are generated.

B. Path-Searching Algorithm

There are two types of nets need to be handled during the refinement. One type is "new" nets that just appear at the current level, as shown by solid lines in Fig. 14(b). These nets are relatively short and do not cross coarser tile boundaries, thus, they are not modeled at coarser levels. We call them *local nets*. New paths need to be created for these nets at the current level. Another set of nets are those carried over from the previous coarser-level routing. We need to refine the tile-to-tile paths for these nets at the current level.

Finding paths for the local nets is relatively easy as they are short and each net crosses two tiles at most. Thus, during each refinement stage, we determine paths for these nets prior to coarser level path refinements. Furthermore, routing these nets before any refinement gives a more accurate estimation of local resources.

The major part of the refinement work comes from refining those coarser-level nets. In general, the amount of work needed for refinement depends on the quality of the coarse level solution. In one extreme, the choices of the paths at the coarser level are also optimal at the current level. We only need to refine the paths within the regions defined by the paths in coarse
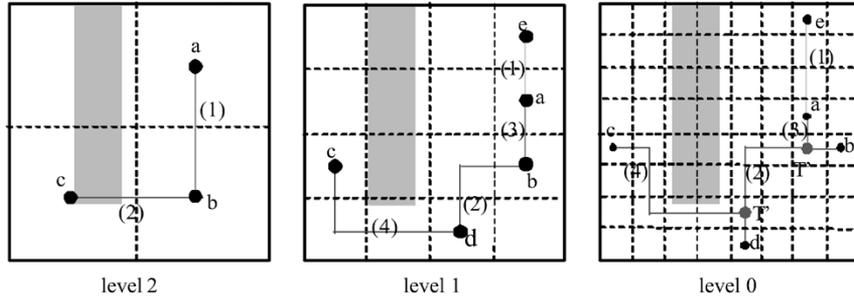
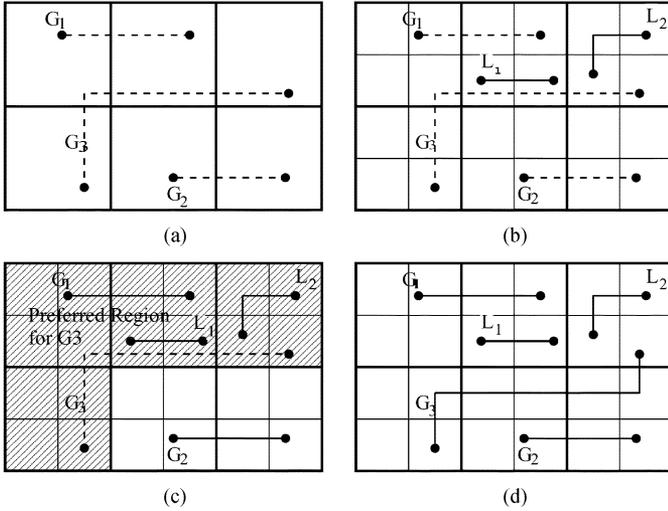Fig. 13.   Gradual construction of a Steiner tree.

Fig. 14.   Constrained maze refinement. Previous routing at coarse grid is shown in (a). Local nets at current level are routed first. Preferred regions are defined by path at coarse tile, as shown in (c). However, not restricted to higher-level results, the modified maze algorithm can change the upper-level tile constraints by seeing the local congestion and arrive at better solutions (d). (a) Coarse routing. (b) Local nets. (c) Constraint region. (d) Final routing.

tiles. In this case, the multilevel algorithm is the same as a hierarchical algorithm. On another extreme, when the coarser solution is totally useless or misleading, the best we can do at the current level is to discard the coarser level solution and search for a new path for each coarse level net among all finer tiles at this level. In this case, we end up doing full planning at the current level. We believe that the reality lies between these two extreme cases. A good coarse level-routing solution provides some good hints as to where the best solution might be. However, if we restrict our search space for the finer tile path to be totally within coarse level tiles planned in the previous level, as in a hierarchical approach, we will lose the flexibility to correct the mistakes made by coarser levels.

In order to keep the coarser-level guide while still maintaining the flexibility to search for all finer tiles, we have implemented a net-by-net refinement algorithm using a modified maze-searching algorithm. We use the path on coarser tiles as a guide to search for a path at the current level. A *preferred region* is defined as the set of tiles that the coarse level path goes through. Weights and penalties associated with each routing graph edge are computed based on the capacities and usage by routed nets. Additional penalties are assigned to graph edges linking to and going between the graph nodes corresponding

to tiles that are not located within the preferred region, as shown in Fig. 14(c). Dijkstra's shortest path algorithm [31] is used to find a weighted shortest path for each net, considering wirelength, congestion, and the coarser-level planning results. In general, Dijkstra's algorithm may be slow in searching for a path in a large graph. However, by putting penalties to nonpreferred regions, we can guide the path to search within the preferred regions first. The penalty is chosen so that it does not prevent the router from finding a better solution that does not fall into the preferred region. Fig. 14(d) shows an example where, with more accurate finer-level tile information and local nets information, the modified maze routing algorithm finds a path for net G_3 that is not totally within its preferred region.

C. History-Based Iterative Refinement

A simple refinement strategy is that the refinement is processed one net at a time in a fixed order only once at each level. This scheme works well when nets are evenly distributed on each level. However, the distribution of nets may not always be smooth. In some designs, a huge amount of local nets would suddenly appear when the routing engine moves to a finer level, and make the refinement problem at that level particularly difficult. Also, all nets are routed one-by-one in each level, adversely affecting the net-ordering problem. Furthermore, once an inferior solution is obtained at a coarser level, more refinement effort would be needed to correct it at finer levels. Limiting to only one round of refinement may not be enough to guarantee satisfactory results.

To handle the problem, in MARS, a history-based iterative refinement similar to one in [7] is applied to each level. The main idea of the history-based method is to iteratively update the edge routing cost with the consideration of historical congestion information and reroute all nets based on the new edge cost functions. The cost of edge e during the ith iteration is calculated by:

$$\text{Cost}(e, i) = \alpha \times \text{congestion}(e, i) + \beta \times \text{history}(e, i) \tag{5}$$

$$\text{history}(e, i) = \text{history}(e, i - 1) + \gamma \times \text{congestion}(e, i - 1) \tag{6}$$

where $\text{congestion}(e, i)$ is a three-tier slope function of the congestion on e, $\text{history}(e, i)$ is the history cost, indicating how congested that edge was during previous iterations, and α, β, γ are scaling parameters.

The congestions of the routing edges are updated every time a path of a net is routed. After each iteration, the history cost of each edge is increased according to (6). Then, the congestion of all edges are scanned to determine whether another iteration is necessary. If so, all edge usages are reset to zero and the refinement process at the same level is restarted.

Multiple iterations at every level may be time consuming when the routing graph is large. We try to control the planning time by the level number. We also make the planning engine iterate more rounds at the coarser levels than at the finer levels to improve the quality and the runtime.

D. Finalization of the Planning Result

With the upward pass refinement process, we can get a globally optimized net planning solution on the finest tiles. Finally, we use a detailed routing engine to find the final connection for each net under the guide of the tile-to-tile paths found by the multilevel planning algorithm. The MARS detailed router is based on the gridless detailed routing engine, DUNE, which is presented in [30].

In MARS, the detailed router will route all nets one-by-one, with each net confined by the planning path corridor generated by the multilevel planning process. The reasons that we choose net-based routing for MARS are: 1) due to the scalable planning capability of MARS, the finest tile size in MARS can be much smaller than the global routing tile in flat global routing methods and 2) the multilevel planning engine works in a net-based fashion, a net-based detailed router will be more consistent with the planning engine. However, DUNE is proposed for tile-based detailed routing, which is different from the net-based routing in MARS. Therefore, several changes are introduced to DUNE:

1) Tile-Based Detailed Routing Grid Graph: Instead of generating one 2-D array grid graph [Fig. 15(a)], as in [30], MARS stores a 2-D grid array at every global routing tile, as shown in Fig. 15(c), which is called *distributed grid graph*. Before a net is routed, the grids of the tiles that the net spans are combined together to generate the detailed routing graph of that net. Since the obstacles outside the corridor are not considered, the size of the resulting routing graph is much smaller than the original grid graph in DUNE. Considering the same net represented by the shadowed tiles in both Fig. 15, in (b), there are six grids in both x and y directions, while in (d), there are only two grids in both x and y directions.

2) Segmentation of the Long Nets: For chip-level global nets, searching for the exact path, even within the planned corridor, is quite time consuming. To make the detailed routing scalable to large designs, long nets will be segmented to a group of short *subnets*, each will be routed individually, shown in Fig. 16(a). The subnets are routed from the source point of the original net to the target point. For the net in Fig. 16(a), the subnets are routed in the order of sn1, sn2, and sn3. Except for the last subnet sn3, the destination of all other subnets, sn1 and sn2, are the connecting boundaries of the current subnet corridor to the next subnet corridor, which are the thick segments T1 and T2, in Fig. 16(a). Once the destination segment is hit, such as the case in Fig. 16(b), the hit point will become the source point of the next subnet, shown as S1 in Fig. 16(c).
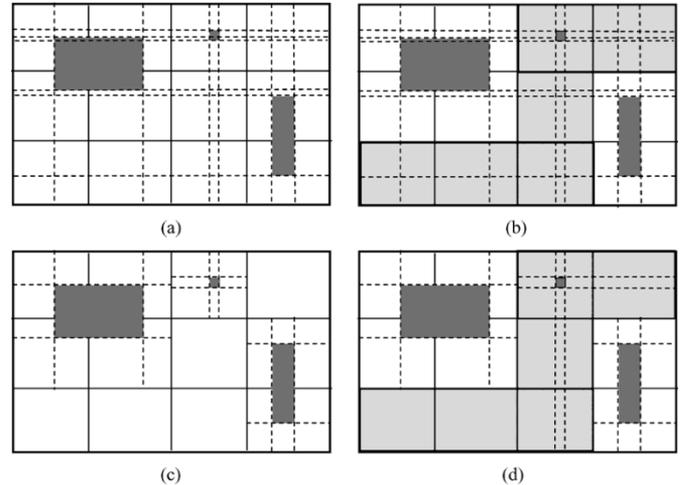

Fig. 15. Distributed connection graph. (a) Original grid graph of DUNE. (b) Connection graph of DUNE. (c) Distributed grid graph. (d) Connection graph from a distributed grid graph.

3) Point-to-Path Routing Engine: DUNE searches for a point-to-point path for each net. In MARS, a point-to-path routing engine is necessary for several reasons: 1) multipin nets are decomposed as Steiner trees, so a point-to-path detailed routing engine is needed to finalize the Steiner trees 2) long two-pin nets are also segmented to short subnets. For many of the subnets, the destinations are not fixed points, but are the boundaries of the routing tiles.

VI. EXPERIMENTAL RESULTS

The multilevel router, MARS, has been implemented, including recursive coarsening with resource reservation, a multicommodity flow-based initial routing algorithm and a history-based iterative refinement. The multilevel routing results are finalized using the efficient multilayer gridless routing engine presented in [30]. To reduce the workload of the detailed router, long nets are cut into short subnets before the detailed routing process. In case of routing failure, further attempt (either enlargement of routing corridor or ripup-and-reroute), will be applied to the unfinished nets.

We have tested our multilevel routing scheme on two sets of benchmarks, uniform design rule (DR) benchmarks with the same wire widths and spacings for all nets at each layer, and variable DR benchmarks, where different nets may have different wire widths and spacings. The benchmarks include MCM examples and several standard cell examples (Table I). Mcc1 and Mcc2 are MCM examples, where Mcc1 has 6 modules, Mcc2 has 56 modules. The original uniform DR circuits are modified to generate the set of variable DR examples. The longest 10% nets are widened to twice the original width, while the next 10% are widened to 150% the original width. The distribution of the two-pin nets at each planning level is shown in Table II. The numbers are the percentage of the two-pin nets that are planned in that level. For most standard cell circuits, there is an average of 35% nets residing within the finest level. Starting from the finest level, the number of planned nets decreases at a steady pace when going from the finer to the coarser levels. The distribution of nets on each level
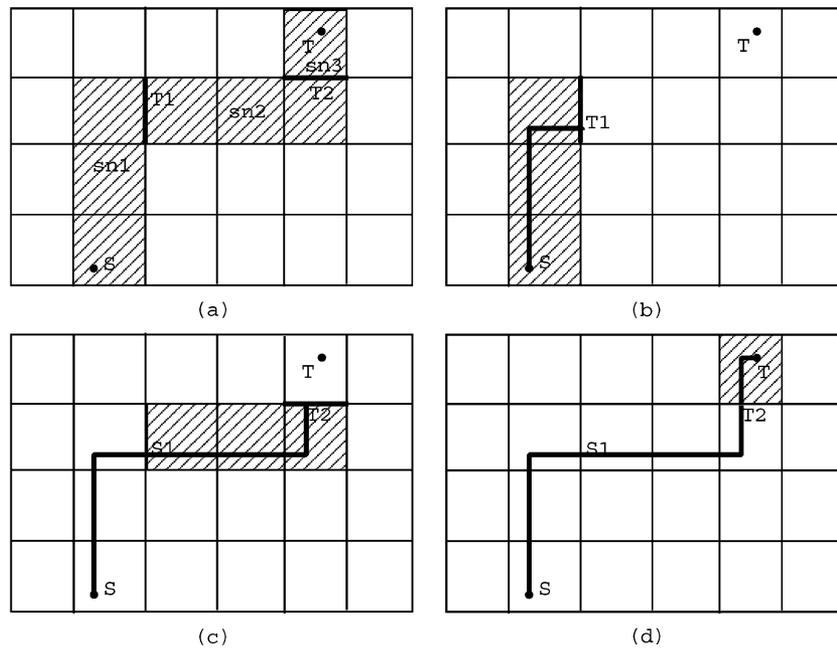
Fig. 16.   Segmentation of a long net. (a) Segment a long net to subnets sn1, sn2, and sn3. (b) Route sn1. (c) route sn2. (d) Route sn3.

TABLE I
EXAMPLES USED FOR MULTILEVEL ROUTING

Circuit	Size(μm)	#Layers	#2-Pin Nets	#Cells	#Pins	#Levels	Division
Mcc1	39000 ×45000	4	1694	MCM	3101	4	100 × 86
Mcc2	152400 ×152400	4	7541	MCM	25024	5	169 × 169
Struct	4903 ×4904	3	3551	n/a	5471	4	68 × 68
Primary1	7552 ×4988	3	2037	n/a	2941	5	104 × 70
Primary2	10438 ×6468	3	8197	n/a	11226	4	72 × 45
S5378	4330 ×2370	3	3124	1659	4734	3	30 × 16
S9234	4020 ×2230	3	2774	1450	4185	3	28 × 15
S13207	6590 ×3640	3	6995	3719	10562	4	48 × 26
S15850	7040 ×3880	3	8321	4395	12566	3	49 × 27
S38417	11430 ×6180	3	21035	11281	32210	3	79 × 42
S38584	12940 ×6710	3	28177	14716	42589	4	92 × 47

TABLE II
NET DISTRIBUTIONS AT EACH LEVEL

Circuit *	Level 0	Level 1	Level 2	Level 3	Level 4
Mcc1	94.2%	91.4%	86.7%	81.5%	0%
Mcc2	100%	100%	100%	99.6%	98.0%
Struct	81.8%	67.7%	48.7%	31.8%	0%
Primary1	93.8%	86.4%	75.2%	52.7%	35.3%
Primary2	82.9%	71.2%	50.4%	31.9%	0%
S5378	67.1%	45.3%	27.5%	0%	0%
S9234	63.5%	42.4%	23.8%	0%	0%
S13207	65.6%	45.6%	28.7%	16.4%	0%
S15850	65.5%	43.5%	25.9%	0%	0%
S38417	63.0%	42.3%	25.9%	0%	0%
S38584	67.3%	44.7%	26.4%	14.7%	0%

is quite smooth. However, for the MCM test cases, most of the nets are planned since the coarsest level.

Our experimental results (Table III) are collected on a Sun Ultra10 440–MHz workstation. "#Total Subnets" are the total 2-pin nets seen by the detailed router. Since long 2-pin nets are segmented to shorter subnets, this number not only depends on the number of multiple pin nets, but also depends on the net planning results. For circuits with variable wire widths and spacings, extra runtime is spent on detailed routing since the detailed routing graph is denser. Table IV shows the percentage of grids visited during refinement that are out of the preferred region. For most standard cell examples, the searching outside the preferred region is usually limited, mostly around or below 20%, depending on how congested the design is. However, for MCM examples with many long global nets, the ratios are higher, which accounts for the longer runtime for these circuits.

TABLE III
ROUTING RESULTS FOR VARIABLE WIRE WIDTHS AND SPACINGS EXAMPLES

Circuit	Uniform DR examples		Variable DR examples	
	#Failed Nets	Run-time(s)	#Failed Nets (#Total Sub-nets)	Run-time(s)
S5378	0	30	1(3687)	70.25
S9234	0	22.8	0	50.1
S13207	0	85.2	0	179.5
S15850	0	107.1	2(10373)	219.4
S38417	0	250.9	0	466.5
S38584	0	466.1	41(34246)	1125.8
Struct	0	31.6	0	36.3
Primary1	0	33.5	0	47.4
Primary2	0	162.7	0	296.7
Mcc1	0	105.9	0	148.1
Mcc2	0	1916.9	27(99715)	3388.8
Avg.		1		1.87

TABLE IV
NUMBER OF VISITED GRIDS OUTSIDE THE PREFERRED REGION AT EACH LEVEL

Circuit	Level 0	Level 1	Level 2	Level 3
Mcc1	37.7%	31.7%	25.4%	
Mcc2	58.2%	48.5%	42.0%	34.7%
Struct	2.9%	3.5%	2.5%	
Primary1	0.6%	0.7%	5.1%	3.00%
Primary2	0.7%	3.5%	3.9%	
S5378	19.4%	24%		
S9234	17.3%	19.1%		
S13207	21.7%	17.4%	21.10%	
S15850	17.2%	20.8%		
S38417	13.8%	21.5%		
S38584	18.5%	15.5%	19.9%	

TABLE V
COMPARISON OF THREE-LEVEL AND MULTILEVEL ROUTING RESULTS

Circuit	3-level Routing		Multilevel Routing	
	#Failed Nets (#Total sub-nets)	Run-time(s)	#Failed Nets	Run-time(s)
S5378	517(3124)	430.2	0	30
S9234	307(2774)	355.2	0	22.8
S13207	877(6995)	1099.5	0	85.2
S15850	978(8321)	1469.1	0	107.1
S38417	1945(21035)	3560.9	0	250.9
S38584	2535(28177)	7086.5	0	466.1
Struct	21 (3551)	406.2	0	31.6
Primary1	19 (2037)	239.1	0	33.5
Primary2	88 (8197)	1311	0	162.7
Mcc1	195 (1694)	933.2	0	105.9
Mcc2	2090 (7541)	12333.6	0	1916.9
Avg.		11.7		1

TABLE VI
EFFECTS OF MULTICOMMODITY FLOW BASED INITIAL ROUTING

Circuit	with MCF initial routing		without MCF initial routing	
	#Failed Nets	Run-time(s)	#Failed Nets	Run-time(s)
vd_S5378	1	70.25	3	69.3
vd_S9234	0	50.1	0	43.7
vd_S13207	0	179.5	7	194.8
vd_S15850	2	219.4	3	224.2
vd_S38417	0	466.5	5	423.9
vd_S38584	41	1125.8	42	1148.6
vd_Struct	0	36.3	0	35.7
vd_Primary1	0	47.4	0	44.4
vd_Primary2	0	296.7	2	297.2
vd_Mcc1	0	148.1	1	142.9
vd_Mcc2	27	3388.8	67	3621.9
Avg.		1.02		1

TABLE VII
EFFECTS OF PLANNING ENHANCEMENTS ON CONGESTION MINIMIZATION

Circuit	maximum congestion after multilevel planning	
	with enhancements	without enhancements
S5378	80%	85%
S9234	78%	80%
S13207	83%	96%
S15850	81%	88%
S38417	80%	86%
S38584	87%	92%
Struct	49%	49%
Primary1	39%	43%
Primary2	36%	45%
Mcc1	80%	80%
Mcc2	79%	85%
Avg.	1	1.08

We compared MARS with the three-level routing flow recently presented at ISPD'2000 [1] using the uniform DR examples. The three-level flow features a performance-driven global router [10], a noise-constrained wire spacing and track assignment algorithm [12], and finally a gridless detailed routing algorithm with wire planning [1]. In this experiment, the global router partitions each example into 16×16 routing tiles. Nets crossing the tile boundaries are partitioned into subnets within each tile. After the pin assignment, the gridless detailed routing algorithm routes each tile one by one. From the results (Table V), we can see that the multilevel routing algorithm helps to eliminate failed nets, and reduce the runtime by $11.7\times$.

Table VI shows the effects of using the multicommodity flow (MCF) based algorithm for initial routing result generation. The set of variable DR benchmarks is showed since the effects of the MCF algorithm is more obvious. Compared to a net-by-net maze routing based approach, the MCF algorithm improves the completion rate with about 2% more runtime on average. The fact that MCF version is even faster for larger and more difficult designs, like S38584 and Mcc2, shows that the flow-based algorithm is effective for relieving local congestion and helps to reduce the workload of finer level planning process. The effectiveness of the two major enhancements for congestion minimization, resource reservation and iterative deletion, is shown in Table VII. The maximum congestions after the multilevel planning process are compared for the planning engine with the enhancements and without the enhancements. Without the enhancements, the maximum congestion is 8% larger in average.

Another approach to handling large designs is to use a hierarchical routing flow followed by a ripup and replan. We have discussed the difference between a multilevel router and a hierarchical router in Section II. We modified our multilevel flow and made it a hierarchical approach. Table VIII compares the routing results of such a hierarchical approach with those of the multilevel approach. Although the hierarchical approach gains a little bit in runtime in some cases, by constraining the search spacing during the uncoarsening process, it loses to our multilevel routing in terms of completion rate. This trend is especially

TABLE VIII
COMPARISON OF HIERARCHICAL ROUTING AND MULTILEVEL
ROUTING RESULTS

Circuit	Hierarchical Routing		Multilevel Routing	
	#Failed Nets (#Total sub-nets)	Run-time(s)	#Failed Nets	Run-time(s)
S5378	1(3390)	22.6	0	30
S9234	0	15.7	0	22.8
S13207	1(7986)	60.9	0	85.2
S15850	1(9587)	75.8	0	107.1
S38417	0	223.2	0	250.9
S38584	3(31871)	334.6	0	466.1
Struct	0	21.6	0	31.6
Primary1	0	34.6	0	33.5
Primary2	0	164.8	0	162.7
Mcc1	377(13338)	205.3	0	105.9
Mcc2	7409(96030)	4433.0	0	1916.9
Avg.		1.04		1

true in designs with many global nets, such as Mcc2, which means that the multilevel planning method can generate planning results with better quality.

VII. CONCLUSION AND FUTURE WORK

We present a novel routing system MARS using a multilevel method. It has several advantages. First, it scales well on larger designs. The partition of routing regions provides a natural hierarchy for the routing levels. The downward pass of recursive coarsening builds the representations of routing regions at different levels. When the designs become larger, additional levels can be added in the multilevel framework, while the overall routing flow is preserved. Compared to a classical hierarchical routing flow and a recent three-level routing flow consisting of a global router, a pin assignment algorithm and a detailed router, MARS provides better routing results in terms of both completion rate and runtime. This is due to its ability of continuous optimization across multiple levels, as well as the ability of integrating different algorithms in the flow.

There are several research directions for further enhancements. One possible improvement is a more clever and systematic way of routing region division and routing graph generation. Both the routing tile size and the number of tiles to be merged during the coarsening process can vary according to the actual circuit design. We tried some ideas of nonuniform merging during the coarsening process. Though we have not yet achieved consistently better results, we still believe that nonuniform coarsening may be a good way to reasonably distribute the computing resource to different regions of the layout according to the actual needs. Also, since the multilevel framework has been successfully applied to partitioning, placement, and routing, it is interesting to investigate if there exists a unified way of integrating these algorithms into a single powerful multilevel optimization flow for VLSI physical design.

REFERENCES

[1] J. Cong, J. Fang, and K. Khoo, "DUNE: A multi-layer gridless routing system with wire planning," in *Proc. Int. Symp. Phys. Design*, Apr. 2000, pp. 12–18.
[2] S. Akers, "A modification of Lee's path connection algorithm," *IEEE Trans. Comput.*, vol. EC-16, no. 2, pp. 97–98, Feb. 1967.
[3] J. Soukup, "Fast maze router," in *Proc. 15th Design Automation Conf.*, 1978, pp. 100–102.
[4] K. Mikami and K. Tabuchi, "A computer program for optimal routing of printed ciurcuit connectors," *IFIPS Proc.*, vol. H-47, pp. 1475–1478, 1968.
[5] D. Hightower, "A solution to line routing problems on the continuous plane," in *Proc. IEEE 6th Design Automation Workshop*, 1969, pp. 1–24.
[6] R. Nair, "A simple yet effective technique for global wiring," *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.*, vol. 6, no. 2, pp. 165–172, Feb. 1987.
[7] L. McMurchie and C. Ebeling, "Pathfinder: A negotiation-based performance-driven router for FPGAs," in *Proc. ACM Symp. Field-Program. Gate Array*, Feb. 1995, pp. 111–117.
[8] R. Carden, J. Li, and C.-K. Cheng, "A global router with a theoretical bound on the optimal solution," *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.*, vol. 15, no. 2, pp. 208–216, Feb. 1996.
[9] C. Albrecht, "Provably good global routing by a new approximation algorithm for multicommodity flow," in *Proc. Int. Symp. Phys. Design*, Mar. 2000, pp. 19–25.
[10] J. Cong and P. Madden, "Performance driven multilayer general area routing for PCB/MCM designs," in *Proc. 35th Design Automation Conf.*, Jun. 1998, pp. 356–361.
[11] J. Cong, L. He, C.-K. Koh, and P. Madden, "Performance optimization of VLSI interconnect layout," *Intergr., VLSI J.*, vol. 21, no. 1–2, pp. 1–94, 1996.
[12] C. Chang and J. Cong, "Pseudo pin assignment with crosstalk noise control," *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.*, vol. 20, pp. 598–611, Mar. 2001.
[13] Semiconductor Industry Association, *International Technology Roadmap Semiconductors*, 2001.
[14] M. Burstein and R. Pelavin, "Hierarchical channel router," in *Proc. 20th Design Automation Conf.*, 1983, pp. 519–597.
[15] J. Heisterman and T. Lengauer, "The efficient solution of integer programs for hierarchical global routing," *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.*, vol. 10, no. 6, pp. 748–753, Jun. 1991.
[16] D. Wang and E. Kuh, "A new timing-driven multilayer MCM/IC routing algorithm," in *Proc. IEEE Multi-Chip Module Conf.*, Feb. 1997, pp. 89–94.
[17] M. Marek-Sadowska, "Global router for gate array," in *Proc. Int. Conf. Comput. Design*, Oct. 1984, pp. 332–337.
[18] Y. Lin, Y. Hsu, and F. Tsai, "Hybrid routing," *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.*, vol. 9, no. 2, pp. 151–157, Feb. 1990.
[19] M. Hayashi and S. Tsukiyama, "A hybrid hierarchical approach for multi-layer global routing," in *Proc. Eur. Conf. Design Test*, Mar. 1995, pp. 492–496.
[20] S. Lin and Y. Chang, "A novel framework for multilevel routing considering routability and performance," in *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.*, 2002, pp. 51–58.
[21] T. Ho, Y. Chang, S. Chen, and D. Lee, "A fast crosstalk- and performance-driven multilevel routing system," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 2003, pp. 382–387.
[22] J. Cong, J. Fang, and Y. Zhang, "Multilevel approach to full-chip gridless routing," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 2001, pp. 396–403.
[23] J. Cong, M. Xie, and Y. Zhang, "An enhanced multilevel routing system," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 2002, pp. 51–58.
[24] A. Brandt, "Multi-level adaptive solution to boundary value problems," *Math. Comput.*, vol. 31, no. 138, pp. 333–390, 1977.
[25] W. Briggs, *A Multigrid Tutorial*. Philadelphia, PA: SIAM, 1987.
[26] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel hypergraph partitioning: Applications in VLSI domain," *IEEE Trans. VLSI Syst.*, vol. 7, no. 1, pp. 69–79, Mar. 1999.
[27] J. Cong, S. Lim, and C. Wu, "Performance driven multilevel and multiway partitioning with retiming," in *Proc. 37th Design Automation Conf.*, Jun. 2000, pp. 274–279.
[28] T. F. Chan, J. Cong, T. Kong, J. Shinnerl, and K. Sze, "An enhanced multilevel algorithm for circuit placement," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 2003, pp. 299–306.
[29] M. Wang, X. Yang, and M. Sarrafzadeh, "Dragon2000: Fast standard-cell placement for large circuits," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 2000, pp. 260–263.
[30] J. Cong, J. Fang, and K. Khoo, "An implicit connection graph maze routing algorithm for ECO routing," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1999, pp. 163–167.
[31] J. Cong, A. Kahng, and K. Leung, "Efficient algorithms for the minimum shortest path Steiner arborescence problem with applications to VLSI physical design," *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.*, vol. 17, no. 1, pp. 24–39, Jan. 1999.
[32] M. Borah, R. Owens, and M. Irwin, "An edge-based heuristic for Steiner routing," *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.*, vol. 13, no. 12, pp. 1563–1568, Dec. 1994.

[33] J. Cong and X. Yuan, "Routing tree construction under fixed buffer locations," in *Proc. 37th Design Automation Conf.*, Jun. 2000, pp. 379–384.

[34] N. Garg and J. Konemann, "Faster and simpler algorithms for multi-commodity flow and other fractional packing problems," in *Proc. Annu. Symp. Found. Comput. Sci.*, Nov. 1998, pp. 300–309.

[35] R. Karp, F. Leighton, R. Rivest, C. Thompson, U. Vazirani, and V. V. Vazirani, "Global wire routing in two-dimensional arrays," *Algorithmica*, vol. 2, pp. 113–129, 1987.

[36] J. Griffith, G. Robins, J. Salowe, and T. Zhang, "Closing the gap: Near-optimal steiner trees in polynomial time," *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.*, vol. 13, no. 11, pp. 1351–1365, Nov. 1994.

[37] C. Chiang, M. Sarrafzadeh, and C. K. Wong, "A powerful global router: Based on Steiner min-max trees," in *Proc. IEEE Int. Conf. Computer-Aided Design*, Nov. 1989, pp. 2–5.

[38] C. Chiang, M. Sarrafzadeh, and C. Wong, "A weighted-steiner-tree-based global router," Manuscript, 1992.

Jie Fang received the B.S. degree in computer science and engineering from Tsinghua University, Beijing, China, and the Ph.D. degree in computer science from University of California, Los Angeles, in 2001.

Since 2001, he has been a Staff Scientist with the Broadcom Corporation, Irvine, CA. His research interests are in physical design of very large scale integration circuits.

Jason Cong (S'88–M'90–SM'96–F'00) received the B.S. degree from Peking University, Beijing, China, in 1985 and the M.S. and Ph.D. degrees from the University of Illinois, Urbana-Champaign, in 1987 and 1990, respectively, all in computer science.

Currently, he is a Professor and Co-Director of the VLSI CAD Laboratory in the Computer Science Department, University of California, Los Angeles. He has been appointed as a Guest Professor at Peking University since 2000. He has published over 170 research papers and led over 20 research projects supported by the Defense Advanced Research Projects Agency, the National Science Foundation, the Semiconductor Research Corporation (SRC), and a number of industrial sponsors in these areas. His research interests include layout synthesis and logic synthesis for high-performance low-power VLSI circuits, design and optimization of high-speed VLSI interconnects, field-programmable gate array (FPGA) synthesis, and reconfigurable computing.

Prof. Cong has served as the General Chair of the 1993 ACM/SIGDA Physical Design Workshop, the Program Chair and General Chair of the 1997 and 1998 International Symposia on FPGAs, respectively, and on program committees of many VLSI CAD conferences, including the Design Automation Conference, International Conference on Computer-Aided Design, and International Symposium on Circuits and Systems. He is an Associate Editor of *ACM Transactions on Design Automation of Electronic Systems* and IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS. He received the Best Graduate Award from Peking University, in 1985, and the Ross J. Martin Award for Excellence in Research from the University of Illinois, in 1989. He received the Research Initiation and Young Investigator Awards from the National Science Foundation, in 1991 and 1993, respectively. He received the Northrop Outstanding Junior Faculty Research Award from the University of California, in 1993, and the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN Best Paper Award 1995. He received the ACM Recognition of Service Award in 1997, the ACM Special Interest Group on Design Automation Meritorious Service Award in 1998, and the Inventor Recognition and Technical Excellence Awards from the SRC, in 2000 and 2001, respectively.

Min Xie (S'02) received the B.E. degree in computer science from Tongji University, Shanghai, China, in 1997 and the M.S. degree in computer science from Tsinghua University, Beijing, China, in 2001. He is currently pursuing the Ph.D. degree in computer science at the University of California, Los Angeles.

His research interests include very large scale integration physical design, placement, and global routing.

Yan Zhang (S'01) received the B.S. degree in computer science and application from Zhejiang University, Hangzhou, China, in 1997 and the M.S. degree in computer science from Tsinghua University, Beijing, China, in 2000. She is currently pursuing the Ph.D. degree in computer science at the University of California, Los Angeles.

Her research interests include physical design of very large scale integration circuits and three-dimensional ICs.