CHIP-LEVEL AREA ROUTING

Le-Chin Eugene Liu Hsiao-Ping Tseng Carl Sechen
Department of Electrical Engineering, Box 352500
University of Washington, Seattle, WA 98195
{liu,hptseng,sechen}@twolf.ee.washington.edu

ABSTRACT

We present a chip-level area router for modern VLSI technologies. The gridless area router can handle any number of layers, as well as rectilinear blockage areas on any layer. A two-stage divide-and-conquer strategy is applied so that the area router can handle very large chips. The first stage includes an area-minimization loop by using an efficient and accurate multi-layer global router. The global router minimizes the chip area while performing the global routing. According to the global routing results, switchboxes are generated for the whole chip area. Then the switchboxes are sent to the second stage for detailed routing, in which a tile-expansion based switchbox router is used. With multi-level rip-up and re-route techniques, the detailed router is shown to be able to complete many difficult switchboxes. The router was tested on the MCNC building block circuits. Our results show better chip areas than the best previously published results.

1. INTRODUCTION

An area-efficient, multi-layer, and gridless chip-level router is presented in this paper. The gridless feature provides the required elements for attacking signal integrity problems, such as crosstalk [4]. A timing optimizer can estimate the performance impact of interconnect and convert the performance requirements into geometric constraints (e.g. increased wire width and larger spacing) for our gridless area router. In this paper, we mainly concentrate on the area routing problem. The previous work in the gridless area routing domain and in the grided area routing domain is briefed as follows.

Most of the area routers are maze-based routers. Without the guidance of a global routing heuristic to evenly assign nets to the routing regions, a pure maze router will usually fail to solve the problem because some of the routing regions are too congested. A maze router is limited to routing nets serially and does not have any sense of the global view. The previous area routing work mainly falls into two groups - regional area routers and hybrid area routers. Regional area routers [5]-[22], which do not solve the net congestion problem, are applicable to single routing region problems such as the channel routing problem, the switch box problem and the cell generation problem. Hybrid area routers [23]-[26], which distribute nets evenly into regions and alleviate the routing congestion problem, are applicable to multiple routing region problems such as chip-level row-based circuits and block-based circuits. The previous work in these two groups are briefly described below.

1.1. Regional Area Routing

Grid based implementations of maze routing algorithms [7][17][20][21] have more flexibility to solve a general routing with obstacles problem than the switchbox routers [5][8][9]. However, the grid based data structure suffers from large data storage complexity and requires search of a huge solution space when facing large circuits. These maze routers route nets serially and therefore need an extensive rip-up and reroute heuristic (usually called a local modification) to reorder the nets in congested regions. The rip-up and reroute heuristics contribute further to the performance problem. Line based approaches [13][14][15] have been proposed to alleviate the storage complexity problem by using a simple line based data structure, which consumes less memory space and reduces the search space. However, the simple path formations based on lines means that an optimal path with minimum bends may not be found. An even more efficient data structure, corner stitching, further reduces memory requirements by using rectangularly shaped tiles to represent layout components and spaces.

Previous approaches [10][16][22] based on the corner stitching data structure enjoy the most flexibility in design rules - variable wire width and variable wire spacing. The tile based routers have the potential to handle a larger problem size than the grid-based routers because of the efficient data structure. Margarino, et. al. [10] first implemented an area router based on the corner-stitching data structure. However, all the routing layers and the contact layer were combined into a single tile plane, which is highly segmented. The breadth first maze router searches for paths on the single tile plane for a restricted wiring style. Tsai, et. al. [16] improved the algorithm by using one tile plane per routing layer. An A* algorithm based maze router was developed to route nets in a restricted routing style. The routing of a multi-terminal net is done by a net-forest technique, which expands the paths from each terminal in parallel toward the center of mass. Dion and Monier [22] improved the maze router by using a bi-directional A* search heuristic. However, the path found by their bi-directional search heuristic is not always optimal. An admissible bi-directional A* algorithm certainly could achieve better solutions.

1.2. Hybrid Area Routing

Previous routing frameworks [6][23]-[26] that include a global routing feature to reduce congestion are called hybrid area routers and are capable of handling chip-level circuits. Hierarchical routers [6][25] route nets on the grid plane in a hierarchical way such that the initial routing path is searched for on a very coarse grid plane. In the next iteration, the routers try to refine the routing path by using a grid plane with half the grid size of the previous iteration. Hierarchical routers have an advantage in using one single router to complete a large routing problem. However, the path assignments in the early routing iterations are based on very rough routing capacity estimations and often cause subsequent routability problems. A mechanism to backtrack path assignments to a coarser grid plane is required when routing failure occurs. Lunow [24] developed a three-stage router - a grid based global router, a track assignment detailed router, and a local modification and rip-up and reroute stage. However, its global router does not consider the congestion problem. Codar [23] is a grid based maze router featuring a global router based on a coarse grid map and a grid based maze router with a fast pattern-based local modification and a recursive rip-up and reroute heuristic [27]. However, the grid-based implementation limits the problem size that it can handle.
We developed an area routing system aiming to solve the chip-level routing problem without the limitations of the previous approaches. An example of the routing problem for chip-level block-based circuits is shown in Figure 1. The functional blocks (building blocks) can have any shape and any size. The routing regions are between blocks and over the blocks. The objective is to complete the routing of a building-block circuit while minimizing its core size.

Figure 1: Chip level area routing problem

To solve the routing problem for block-based circuits, we built a routing system featuring three stages - Steiner tree based global routing, switchbox generation, and tile expansion area routing. The program flow is shown in Figure 2. Given a placement of macro cells (or building blocks), the chip area is divided into small regions by cut lines which are the extension lines of the boundaries of the blockage areas (usually the cells) on all layers. If two cut lines are too close, they merge. To construct the global routing graph, we place a node on each layer in every region. The nodes are connected via edges, horizontal edges, and vertical edges. The via edges connect nodes on different layers. The horizontal/vertical edges connect the nodes on the layer used for horizontal/vertical routing. Inside the blockage areas, there are no connecting edges between the nodes. In addition, the cutting edges across the boundary between the blockage and outside areas are directed. Those edges can only be used for the signals to exit the blockage areas. An example is shown in Figure 3.

Figure 2: Flow of our routing system

2. GLOBAL ROUTING/AREA MINIMIZATION

Our area routing algorithm can be divided into two stages --- global routing and detailed routing. For the global routing, we extended the global router introduced in [1]. The global router is for multiple routing layers with rectilinear blockage areas. Typically, the blockage areas are the macro cells (or building blocks) in the layout. Other examples are pre-routed areas or reserved areas on the higher routing layers.

The routing architecture is constructed by first dividing the chip area into small routing regions. Then, a global routing graph is constructed. The global route for each net is searched for on the graph under certain criteria by using an improved Steiner tree heuristic. We next briefly review the routing architecture, and then describe the crucial modifications to the global router in [1] that were needed for its use in chip-level area routing.

2.1. Routing architecture

A brief review of the routing architecture is presented here. Given a placement of macro cells (or building blocks), the chip area is divided into small regions by cut lines which are the extension lines of the boundaries of the blockage areas (usually the cells) on all layers. If two cut lines are too close to each other, the two cut lines are merged. To construct the global routing graph, we place a node on each layer in every region. The nodes are connected by via edges, horizontal edges, and vertical edges. The via edges connect nodes on different layers. The horizontal/vertical edges connect the nodes on the layer used for horizontal/vertical routing. Inside the blockage areas, there are no connecting edges between the nodes. In addition, the connecting edges across the boundary between the blockage and outside areas are directed. Those edges can only be used for the signals to exit the blockage areas. An example is shown in Figure 3. This example is for two routing layers, in which the first one is for horizontal and the second one is for vertical routing. The first layer is not available inside the cells for routing. On the other hand, the second layer (vertical layer) is available inside the cells, i.e., it does not have any blockage areas. The cut lines are shown as dashed lines in the example. The nodes are the small solid squares. The 3-dimensional global routing graph is also shown in the example.

2.2. Chip size estimate

The global router uses two directed graphs to accurately estimate the chip size. The chip’s height and width are calculated the same way. We use the calculation for the chip’s height as an example. First, nodes are placed on the top and bottom sides of the cells. Therefore, a rectangular cell has one node on the top boundary and one node on the bottom boundary. For non-rectangular rectilinear cells, there may be more than one top or bottom boundary. We place one node on each horizontal boundary segment. In addition, we have one source node on the top and one sink node at the bottom of the chip. The height graph is directed from the source node to the sink node. Since the routing model divides the chip area into rectangular regions, we have columns, which may consist of a series of regions, between the cells or between the cells and the source/sink node. In Figure 4(a), the columns are shown by the directed arcs. To simplify the graph, only one directed edge is needed to represent the columns between any pair of nodes. The weight of the edge is determined by the highest column of the columns corresponding to the edge. The height of a column is decided by the horizontal layer that requires the most space. Figure 4(b) shows the final height graph. For illustration purposes, we placed the edges into the highest columns between the nodes, if there is more than one column.
However, they actually represent a set of columns. Inside the cells, each node on the top has an edge to each node at the bottom. The weight of such an edge is the distance between the boundaries. For a rectangular cell, it is the height of the cell. The longest path from source to sink determines the height of the chip.

![Diagram of column and height graphs](image)

Figure 4. graphs for the example in Figure 3

The width is calculated the same way. The only difference is that the edges go horizontally. The estimated chip area is the product of the chip height and width.

### 2.3. Area minimization routing

To minimize the chip area, we try to reduce the height and the width of the chip sequentially. The size of the chip is decided by the longest paths in the height and width graph. The longest path corresponds to the sum of the sizes of a series of routing regions and some cells. Since the size of the cells is fixed, the only possible way to change the size of the longest path is to change the size of the routing regions. The size of the routing regions is determined by the number of routes using the regions. Those routing regions in the longest path are called “critical regions”, which are similar to the “dominant tiles” in [30]. We re-route those nets that use the critical regions during the area minimization routing.

To reduce the height and width of the chip, we re-route the nets in the critical regions to see if there is a new route that can reduce the size of the critical regions. In particular, we dynamically set the weights of graph edges in critical regions and reroute the nets. An edge’s weight is set to infinity if the use of the edge does not reduce the size of the critical region. If such a new route exists, the new chip size is calculated according to the route change. If the chip size is reduced, the new route is accepted, otherwise it is rejected. The algorithm is as follows:

```c
1. for (all nets in the critical regions) {
2.     try to find a new route which reduces the size of the critical regions
3.     /* Set the weights of the edges in critical regions to infinity because the use of those edges don’t reduce the size of critical regions */
4.     if (no such a new route exists) continue
5.     if (the new route does not increase the weight and reduces the chip size)
6.         accept the new route and continue
7.     if (the new route increases the weight)
8.         put the new route in a priority queue according to the increase of the weight
9. }
10. for (each new route in the priority queue in order)
11. if (the new route reduces the chip size)
12.     accept the new route
```

The re-calculation of the chip size is key to the effectiveness of the algorithm, because a new route which reduces the size of critical regions may not reduce the chip size. Besides, new critical regions can be generated when a new route reduces the original critical regions. The above algorithm has another feature. It minimizes the wire length increase while minimizing the chip area. The re-routing process can also be repeated many times to achieve better results. A greedy approach is to repeat it until no further size reduction is possible, i.e. no new route is found or accepted during the iteration.

### 2.4. Area minimization loop

The above area minimization routing is based on the assumption that the macro cells can be pushed away from each other to accommodate more tracks or pushed closer to reduce the required area. This assumption is true for one dimension but not for both dimensions. After the width is reduced, the placement has been changed. It may not be possible to reduce the height as previously assumed. In addition, to reduce the chip area, the placement has to change according to the routing results. The placement change may affect the global routing. The original global routing results may lose some accuracy. From the above analysis, an iterative process is apparently needed for solving the problem. We used the TimberWolf chip area compactor to work with the global router to perform the area minimization.

After the area minimization routing, the global router can estimate the required routing area between the cells. This information is passed to the compactor. The compactor moves the macro cells to minimize the chip area. We do not allow the compactor to move the cells far away from their original positions. The topological relations between the cells are retained during the process. However, the original placement may have changed, and the global routing graph may have changed as well. In such a case, a new area minimization routing is performed. The area minimization routing and compaction process is repeated until the chip area reaches a stable point. The whole algorithm can be demonstrated as follows:

1. The global router performs area minimization routing.
2. The compactor moves the cells.
3. If the area does not converge, go to step 1.
4. The global router performs congestion removal routing.

When the chip area change between the iterations is smaller than a certain percentage of the whole chip area, the convergence condition is satisfied. According to the experimental results, the convergence usually occurs within five iterations. Placement is usually done under the minimum-wire-length condition. If the placement were done under the same minimum-area condition, the convergence would happen faster. When the chip area converges, the final placement is fixed. The macro cells can not be moved, because this is the placement for minimum chip area. Hence, the last step of the algorithm is to solve any possible congestion problems.

### 3. GBOX GENERATION

#### 3.1. Introduction

The global router decides which regions a net goes through. The global routing results provide routing information for each routing region. For each routing region, there is a list of nets that require tracks in the region. We also know the directions that a net is coming from or going to in a region. Since the three-dimensional global routing graph mimics the multiple routing layers, the knowledge of layer usage for each net in each region is also included in the global routing results. Since we know the nets on the boundary between two routing regions, we only need to know the exact positions of the nets on the boundary in order to construct a switchbox. For multi-layer routing, the routing regions are more general than the traditional switchboxes. The routing regions can have pins and blockage areas anywhere in the region. We call such a rectangle with fixed pins and blockage areas a **Gbox** (global routing box). Each routing region corresponds to a Gbox. There is a constraint between adjacent Gboxes, namely that the pin locations on the common boundary have to be consistent. Otherwise, the continuity of the signals will be lost. Figure 5(a) shows the notion of Gboxes, and two of them are shown in the callout. The pin locations on the common boundary are exactly matched. Figure 5(b) shows an
example when the routing is completed inside the two Gboxes.

![Image](a) before routing (b) after routing

**Figure 5. Illustrations of Gbox routing**

### 3.2. Basic concepts

A few terms are defined here. First, we divide a route into several segments. A segment is the maximum continuous routing region span of a net’s global route on a column or a row. Figure 6 shows an example. The dashed lines are cut lines. The thin solid lines are part of the routing graph. A global route and the cell boundaries are shown as thick solid lines. There is a two-pin net, \( NetA \), between two cells. The pins are shown as dots, and the nodes that the pins are mapped to are shown as solid squares. \( NetA \) is routed through a series of regions. The optimal condition is that \( NetA \) has two horizontal tracks, one vertical track, and two vias. Therefore, the route consists of three segments. Segments A and C span two regions horizontally, and Segment B spans three regions vertically. According to the definition of a segment, all the global routes are decomposed into a set of segments.

A segment that contains one or more pins is called a pin segment. For example, in Figure 6, segments A and C are pin segments. If a net passes through a routing region, the net is called a pass-through net in that routing region. For example, segment B of \( NetA \) spans three regions: \((i,j-1), (i,j)\), and \((i,j+1)\). However, it only passes through the middle region, \((i,j)\). Therefore, in region \((i,j)\), \( NetA \) is a pass-through net.

The task for generating a Gbox is to assign the exact positions for each pin in the routing region. The exact positions are represented by the pins of a Gbox. A pass-through net of a routing region crosses the opposite boundaries of the routing region. Therefore, a pass-through net requires two pins on the opposite boundaries of the corresponding Gbox. A Gbox is complete when all the nets using that routing region have pins completely assigned. The detailed router will decide which actual tracks are used inside a Gbox.

![Image](Figure 6. Example of a route and its segments)

Two requirements must be satisfied for constructing the Gboxes. The first one is retaining signal continuity. The second one is making the Gbox feasible for detailed routing, i.e., all nets have to be connected without design rule violations. To satisfy the first requirement, we make sure that a signal’s pins have the same position, including the layer usage, on the shared boundary of two adjacent Gboxes. This is also the first constraint for Gbox generation. The second requirement partially depends on the accuracy of the global routing. Since the global router is good at estimating the routing resources needed, in the Gbox-generating stage, we focus on minimizing the required routing resources for every Gbox.

In order to minimize the routing resource usage, we use two essential concepts for assigning the pin positions. First, our algorithm keeps the segments as straight as possible. If at all possible, a segment is mapped to a single track. It means that a segment has the same pin positions in all the Gboxes that the segment spans. The implementation of this concept is to assign the same pin positions on the opposite boundaries for the pass-through nets. In Figure 7, (a) is the optimal result and (b) is a worse result. The non-optimal cases happen when the corresponding position on the opposite side is not available.

![Image](Figure 7. Connection examples)

Second, our algorithm reduces the number of jogs needed in detailed routing. In Figure 7(c), net A has three fixed pins, and net B has two floating pins. If the pin of net B on the right side falls into the interval between the dashed lines, the pin of net B on the bottom side will have to be on the right side of net A’s vertical track. Otherwise, an extra vertical track (jog) would be needed for routing net B as illustrated in Figure 7(d). Our program tries to assign net B’s pin on the right side to a position that is out of the interval. Therefore, the assignment for net B’s pin on the bottom side can have more freedom.

### 3.3. Segment-positioning algorithm

Based on the above two concepts, we developed the algorithm to generate the Gboxes. Unlike some other Gbox-by-Gbox (region-by-region) algorithms, our algorithm is based on segments. A segment-based algorithm yields better global optimization on the routing resource usage.

The position of a segment is first decided in one of the routing regions that the segment spans. The same position then propagates throughout the whole segment. From the global routing, every routing region has a list of nets that need to be routed in the region for each layer. The program assigns the positions for the nets sequentially in a region. The order of the nets is decided according to the features of the corresponding segments. Here is the algorithm:

1. for (each net) { 
   2.   assign positions for pin segments 
   3. } 
4. build priority queue for the routing regions 
5. build segment preference data 
6. for (each region in the priority queue) { 
   7.   assign positions for the pass-through nets with preference 
   8.   assign positions for the non-pass-through nets with preference 
   9.   assign positions for the pass-through nets without preference 
10.   assign positions for the non-pass-through nets without preference 
11. }

A net’s pins on the macro cells have fixed positions. Since a pin segment contains pin(s) of the cells, it is reasonable to assign the pin segments first. Assigning a segment means deciding the positions of the Gboxes’ pins on the Gbox boundaries within the span of the segment. Our algorithm always tries to assign a straight-line position for a segment unless there is a conflict.

The next step is to assign the segments in the most difficult region for routing. We have a cost function to determine the sequence in which the regions are processed.

\[
\text{Cost} = \text{number of potential jogs} + \text{congestion condition}
\]
It is obvious that a crowded region should be processed before a sparse region because the routing resources are more constrained. The congestion condition in the cost function is equal to 2 for an over-congested region, and 1 for a full region. An over-congested region has more tracks than it can accommodate. A full region has exactly the same number of tracks as its capacity allows. The congestion condition is 0 for all other regions. Another important factor is the number of potential jogs. After Line 3, only the pin segments are assigned. Many of the segments are still floating. If we can identify where the potential jogs may occur and assign the segments properly to prevent jogs from happening, the over-all routing resources needed will be reduced. When the pin segments are assigned, we have fixed pins in the Gboxes that the pin segments span. If two pins belonging to two nets on the opposite boundary have conflicting positions, a jog may be needed. The potential jogs are counted and included in the cost function. If there is a tie for the cost, the number of floating tracks is used to break the tie. That is, a region with more floating tracks has higher priority. Line 4 uses the above cost function to decide the order in which the regions are to be processed.

In the loop of lines 6-11, the regions are processed one by one. Processing a region is to assign the position for each net in the region. The program first assigns a net to a position in a region. Then, the program assigns the same position to the whole segment unless there is a conflict. Therefore, the position has to be decided by the features of the whole segment and not by the relations of the nets in a local region.

One important feature of the segments is the preference data. The preference data is used to reduce the total wire length and the number of potential jogs when assigning segments. The preference of a horizontal segment is decided by the number of the vertical segments connected to the horizontal segment. Similarly, the preference of a vertical segment is decided by the number of the horizontal segments connected to the vertical segment.

Figure 8 shows three examples for a vertical segment. The box in the examples is the region that is currently being processed. In (a), there are two segments connected to the left side and one segment to the right side. The preference is therefore to place the vertical segment near the left side of the Gbox. In (b), there is no preference for the vertical segment. In (c), the vertical segment prefers the right side because there are more horizontal segments connected on the right side.

Figure 8. Segment preference

Line 5 calculates the preference for the segments. The preference data are used in the loop in Lines 6-11. In the loop, those nets (segments) with preference are processed before those nets without preference. The nets are divided into two groups according to the preference data. Among each group, the pass-through nets are assigned first. Pass-through nets need two pins on the opposite boundaries. The positions of the two pins have to be the same in order to have a straight line connection. The more restricted nets are processed first so that they can have more freedom.

After all regions have been processed, the Gboxes are generated according to the assigned positions. For a Gbox, the cells’ pins and blockage areas are placed first. Then, the cross points (pins) on all the boundaries are placed to complete the Gbox. The Gboxes are then passed to the detailed router. The detailed router determines the final routing inside the Gboxes.

Kao [31] developed a switchbox crosspoint assignment (CPA) heuristic for two-layer gate-array gridded routing. It features an iterative global re-routing during CPA stage when the linear assignment CPA heuristic cannot resolve overcongestion. Since our global router accurately estimates routing capacities, it is not necessary to apply global re-routing during the CPA stage. Kao’s system was limited to two-layer routing, was not tested on macro cell circuits, and did not address area reduction.

4. DETAILED ROUTING

Gbox detailed routing is accomplished by a robust tile-based area router which has a multi-level rip-up and reroute strategy and a routing window control to improve the routability. We describe the framework of the tile-based router in section 4.1, and the multi-level rip-up and reroute strategy in section 4.2. Heuristics to decide the routing order and the routing window control are presented in section 4.3 and section 4.4, respectively.

4.1. The Tile-based Area Routing Framework

Our detailed area router is based on the corner-stitching data structure. The corner-stitching data structure has advantages in retrieving the local geometric information and the space interval information. It was introduced by Ousterhout [28]. Tiles on the plane are stitched and combined either into strips of maximal horizontal extent called a horizontal (H) tile plane or strips of maximal vertical extent called a vertical (V) tile plane.

A special technique to mirror the coordinates of an H tile plane at 45 degrees can produce a dual V tile plane. The coupled H and V tile planes are called the Dual Tile Plane (see Figure 9(b), (c)). A V tile plane on coordinate (x, y) is actually an implementation of an H plane on the reflected coordinate (y, x). The maximal H (V) strip tiles on the H (V) tile plane allow fast tile expansion in the H (V) direction. It extends the horizontal routing efficiency of the conventional single tile plane to both directions of tile expansion. We explain the tile expansion patterns and the searching algorithm in the following paragraphs.

Our tile expansion router essentially takes two routing patterns - a metal expansion model and a via expansion model to find a feasible route with minimum cost using an admissible A* algorithm [29]. The metal expansion model is applied when the routing path is expanded on the current metal layer. The via expansion model is applied when the routing path is expanded to the adjacent metal layer. Our expansion models allow unrestricted layer routing - H to V, H to H, V to H and V to V. The cost function weights expanded components, such as a space tile, via, or metal wire, differently. Our algorithms also extend the search path through existing solid tiles with different signals. An efficient rip-up and reroute heuristic has been developed to clean up overlapped wires.
4.2. Multi-level Rip-up and Reroute

The rip-up and reroute (RR) algorithm allows the expansion path to overlap with a non-equivalent solid tile which has a different signal than the current net being routed. To form the routing path on the selected non-equivalent solid tile, the center of its original shape is used as the alignment to shape the new routing path and to meet design rules. By applying this overlapped tile expansion feature, a feasible optimal route is always found with a minimum of overlapping with non-equivalent solid tiles by the maze router. The router first rips up those two-pin nets which own the overlapped non-equivalent solid tiles, draws the new route for the current net, then continues this rip-up and reroute process for those ripped up overlapped nets. A multi-level control mechanism developed to optimize this process is described in subsection A. Subsection B shows the difference and the improvement of our new rip-up and reroute strategy from previous work.

A. Multi-level Rip-up and Reroute

To prohibit nets from being cyclically thrashed out by each other in congested Gboxes, a multi-level control mechanism was implemented. The level of rip-up and reroute is defined as the number of subsequent times that the rip-up and reroute procedure will be executed recursively. A feasible route found at a level i RR process is denoted as a tentative route. A tentative route in the level 0 RR process is not allowed to overlap with non-equivalent tiles, since no subsequent rip-up and reroute executions are available to remove overlaps. The level n RR process for node p is accomplished by the following procedures - (1) find a tentative route, (2) remove nets which own the overlapped tiles on the route, (3) draw the route, (4) run a level n-1 RR process for the removed nets, (5) if any net in step 4 fails, collect the failed solid tiles owned by the failed nets, and (6) call procedure 1 with n = n-1 and with the failed solid tiles not being allowed to be expanded.

A multi-level rip-up and reroute tree (MRRT) is constructed to show the hierarchical relation of nodes in different RR levels (see Figure 10). Net p is processed in the level n RR process and is permitted to find Rt possible (overlapped or non-overlapped) tentative routes. Tentative route rt of net p has Nrt overlapped nets. The mth overlapped net (m ∈ [1, Nrt]) on the route rt of node p is denoted as net(rt,m) in the level n RR process. If net i is prior to net j and net i is on the path to net j in the MRRT, net i is an ancestor to net j and net j is a descendant to net i in the RR process. A tentative route with a large Nrt tends to be more costly due to the more overlapping with existing nets and usually has a longer path than a tentative route with a small Nrt. In order to control the computation time and to obtain better results, a limited number of tentative routes (Rt=4) are considered for each net at each level of the RR process. In our experiments, most of the Gboxes need only a level 0-2 RR process.

B. Comparison with Other Recursive Strategies

The recursive rip-up and reroute strategy has been implemented by Poirier [27] for the cell generator EXCELLERATOR (EXCEL in short). Instead of resolving the conflicts (overlapping wires) after completing the current route of net a, EXCEL tries to resolve the conflict on grid k on the expansion tree whenever grid k is owned by an existing net (say v). The rip-up and reroute process for grid k is recursive and the traversed nodes of the top-level net a are frozen and treated as obstacles. The frozen nodes unnecessarily block potentially better routes, since not all the traversed nodes will appear on the final optimal path of net u. The attempt to immediately resolve the conflict of every overlapping node is very expensive, since not every overlapping node will appear on the final optimal path of the top-level net. His implementation is analogous to our MRRT at Rt=1 and only searches a subset of our routing solution space (see Figure 11). Another drawback of the immediate conflict resolution is that each conflicting node at the topmost level is resolved independently of other resolving attempts. That means the new path found for each conflicting net may be overlapped with the new path of some other conflicting net. There is no guarantee that those conflicting nets on the optimal path of the topmost net can all be successfully rerouted.

4.3. Net Ordering and Routing Enhancements

In a chip-level circuit, most nets are long and go across multiple Gboxes. We observe that most wires go straight without jogs. In a congested region, a jog may cause a ripple effect to push neighboring wires away and therefore degrade routability. To maximize straightened wires, nets with their two pins at opposite boundaries and at the same x or y position are routed first using zero bends as shown in Figure 12(i).

Figure 11: Call graph of Poirier’s recursive approach

B. Comparison with Other Recursive Strategies

The recursive rip-up and reroute strategy has been implemented by Poirier [27] for the cell generator EXCELLERATOR (EXCEL in short). Instead of resolving the conflicts (overlapping wires) after completing the current route of net a, EXCEL tries to resolve the conflict on grid k on the expansion tree whenever grid k is owned by an existing net (say v). The rip-up and reroute process for grid k is recursive and the traversed nodes of the top-level net a are frozen and treated as obstacles. The frozen nodes unnecessarily block potentially better routes, since not all the traversed nodes will appear on the final optimal path of net u. The attempt to immediately resolve the conflict of every overlapping node is very expensive, since not every overlapping node will appear on the final optimal path of the top-level net. His implementation is analogous to our MRRT at Rt=1 and only searches a subset of our routing solution space (see Figure 11). Another drawback of the immediate conflict resolution is that each conflicting node at the topmost level is resolved independently of other resolving attempts. That means the new path found for each conflicting net may be overlapped with the new path of some other conflicting net. There is no guarantee that those conflicting nets on the optimal path of the topmost net can all be successfully rerouted.

4.3. Net Ordering and Routing Enhancements

In a chip-level circuit, most nets are long and go across multiple Gboxes. We observe that most wires go straight without jogs. In a congested region, a jog may cause a ripple effect to push neighboring wires away and therefore degrade routability. To maximize straightened wires, nets with their two pins at opposite boundaries and at the same x or y position are routed first using zero bends as shown in Figure 12(i).

Figure 12: Nets are routed using minimum bends and an additional jog (numbers shown are the bend counts)

1) Route net n with max_bend(n)=0, branch_max(n)=L
2) Route net n with max_bend(n)=1, branch_max(n)=L
3) Route net n with max_bend(n)=2, branch_max(n)=L
4) Route the failed nets in (2) with max_bend(n)=3, branch_max(n)=L
5) Route the failed nets in (1) with max_bend(n)=4, branch_max(n)=L
6) Route the failed nets in (3) with max_bend(n)=4, branch_max(n)=L
7) Route all the failed nets with max_bend(n)=4, branch_max(n)=L
8) Route all the failed nets with max_bend(n)=oo, branch_max(n)=oo

Figure 13: Net routing ordering

In some of the tested benchmark circuits, we encountered very large switchboxes. Several pruning and trimming techniques were developed to route Gboxes as large as 2000x2000 tracks. Since our switchbox generation algorithm creates very few vertical constraints, most nets are connected using small numbers of bends as shown in Figure 12(i)-(iii). The maximum number of bends to route net n is denoted as max_bend(n). If existing wire overlappings block the minimum bend solution, a minimum bend plus one-dog-leg solution may be feasible for most of the cases as shown in Fig.
In a tile plane of $N \times N$ tracks (representing a Gbox), a tile may generate as many as $N$ nodes to the adjacent layers. Essentially, most of these nodes can be pruned in finding the minimum bend solution and the minimum bend plus one-jog solution. The bend control limit provides an upper bound of the tile expansion cost. Any routing solution using the required number of bends is accepted. For each via expansion, only at most the $L (\text{branch}_\text{max}(n))$, e.g. 10) least cost nodes are selected among all the possible generated nodes and the rest of the generated nodes of this expansion are deleted. The net routing order is described in Figure 13. Note that from step 1 to step 7 only non-overlapping routes are allowed and overlapping routes using rip-up and reroute are allowed only in step 8.

4.4. Routing Window Control

Whenever a net fails to be routed inside a routing region, we increase the size of the routing window to accommodate the failed net in the neighboring routing regions. As shown in Figure 14, there is a routing window increasing pattern for each one of six routing failures. Assume in region $R$, the failed nets are $\{n_1, n_2, \ldots\}$ and their window-increasing patterns are $\{W_1n_1, W_2n_2, \ldots\}$. The window-increasing control for region $R$ is then to increase the window size by one Gbox for each of the directions represented in $W_R = (W_1n_1 \cup W_2n_2 \cup \ldots)$. Our router then routes the failed nets inside the increased region $W_R$. If the window-increasing effort $W_R$ doesn’t completely route all the failed nets, the window-increasing control is applied recursively (by one more Gbox in each of the relevant directions) until all nets are completely routed. To control computation time, the recursion is limited to 2 levels. Our observation is that any resulting failed nets require floorplan modification and cannot be resolved manually.

5. RESULTS

We tested our area router on the MCNC building-block benchmark circuits. We used five benchmark circuits: hp, apt, xerox, ami33, and ami49, since those circuits have been used previously [2][3]. In [2], it was shown that TimberWolfMC (the placer and global router) and TimberWolfDR (the detailed router) outperformed all other placement and routing tools at that time. [3] showed that the Branch-and-Bound placement algorithm could get even better results than TimberWolfMC. A commercial router was used to finish the routing. In our tests, the placement was done by TimberWolfMC Version 3.1. It is basically the same placer used in [2]. So we got about the same quality placement as those in [2]. However, our new area router is able to minimize the chip area and very effectively finished the tight space routing. Although our placement may not be as good as those done by the Branch-and-Bound algorithm, the final chip areas are better. Table 1 shows the chip area comparison results. Our results are better than the Branch-and-Bound results for every benchmark circuit. Especially for the largest circuit, ami49, which has 49 blocks and 408 nets, the area is reduced by 6.8%. The CPU time for global routing and

<table>
<thead>
<tr>
<th>Circuit</th>
<th># of Gboxes</th>
<th># of 2-pin nets in Gboxes</th>
<th>before merging, in terms of # of Gboxes</th>
<th># of 2-pin nets in Gboxes</th>
<th>after merge</th>
</tr>
</thead>
<tbody>
<tr>
<td>hp</td>
<td>204</td>
<td>&gt; 1,460</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>apte</td>
<td>152</td>
<td>&gt; 2,170</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>xerox</td>
<td>217</td>
<td>&gt; 920</td>
<td>97.24% (6 failed)</td>
<td>99.72% (6 failed)</td>
<td>100%</td>
</tr>
<tr>
<td>ami33</td>
<td>681</td>
<td>&gt; 2,570</td>
<td>97.06% (20 failed)</td>
<td>99.07% (24 failed)</td>
<td>100%</td>
</tr>
<tr>
<td>ami49</td>
<td>2,856</td>
<td>&gt; 13,150</td>
<td>99.47% (15 failed)</td>
<td>99.86% (18 failed)</td>
<td>100%</td>
</tr>
</tbody>
</table>

Table 2. Completion rate of Gbox routing

Figure 15(a) shows the routing result of circuit ami49 using two metal layers. Our area router is capable of handling any number of layers. To demonstrate this capability, we changed the technologies used for the benchmark circuits from two layers to four layers, and we applied our router to the same circuits again. Figure 15(b) shows the four-layer routing result of circuit ami49. Table 3 shows the area reduction results by changing the technologies from two layers to four layers.

The area reductions are limited by the fact that, even for two layers, the chip areas are dominated by active cell area. Circuit ami33, however, had a large 21% area reduction since it had a relatively large amount of routing space for two layers.
Table 3. Area reduction results

<table>
<thead>
<tr>
<th></th>
<th>2-layer</th>
<th>4-layer</th>
<th>reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>hp</td>
<td>1647 X 3402</td>
<td>1647 X 3402</td>
<td>0.0%</td>
</tr>
<tr>
<td>apt</td>
<td>3888 X 3228</td>
<td>3888 X 3228</td>
<td>0.0%</td>
</tr>
<tr>
<td>xerox</td>
<td>4708 X 4003</td>
<td>4708 X 4003</td>
<td>0.0%</td>
</tr>
<tr>
<td>ami</td>
<td>1311 X 1368</td>
<td>1311 X 1368</td>
<td>0.0%</td>
</tr>
<tr>
<td>ami</td>
<td>7029 X 6582</td>
<td>7029 X 6582</td>
<td>0.0%</td>
</tr>
</tbody>
</table>

6. CONCLUSION

We presented a chip-level area router for modern VLSI technologies. The gridless area router can handle any number of layers, as well as rectilinear blockage areas on any layer. A divide-and-conquer strategy is applied so that the area router can perform routing on a very large chip area. The first stage includes an area-minimization loop by using an efficient and accurate multi-layer global router. The global router minimizes the chip area while performing the global routing. According to the global routing results, Gboxes are generated for the whole chip area. Then the Gboxes are sent to the second stage for detailed routing by means of a tile-expansion based router. With multi-level rip-up and re-route techniques, the detailed router is able to complete many difficult Gboxes. The router was tested on the MCNC building block circuits. Our results show better chip areas than the best previously published results.

REFERENCES