FastRoute 2.0: A High-quality and Efficient Global Router

Min Pan, and Chris Chu
Department of Electrical and Computer Engineering
Iowa State University, Ames, IA 50011
Email: {panmin, cnchu}@iastate.edu

Abstract— **Because of the increasing dominance of interconnect issues in advanced IC technology, it is desirable to incorporate global routing into early design stages to get accurate interconnect information. Hence, high-quality and fast global routers are in great demand. In this paper, we propose two major techniques to improve the extremely fast global router, *FastRoute* [8] in terms of solution quality : (1) monotonic routing, (2) multi-source multi-sink maze routing. The new router is called *FastRoute* 2.0. Experimental results show that *FastRoute* 2.0 can generate high-quality routing solutions with fast runtime compared with three state-of-the-art academic global routers *FastRoute*, *Labyrinth* [9] and *Chi Dispersion* router [10]. On the set of benchmarks used in [8] and [10], the total overflow of *FastRoute* 2.0 is 98, compared to 1012 (*FastRoute*), 2846 (*Labyrinth*) and 1271 (*Chi Dispersion* Router). The runtime of *FastRoute* 2.0 is 73% slower than *FastRoute*, but 78× and 37× faster than *Labyrinth* and *Chi Dispersion* router. The promising results make it possible to integrate global routing into early design stages. This could dramatically improve the design solution quality.**

I. Introduction

As feature size in advanced VLSI technology continues to shrink, interconnect delay has become the dominant factor in circuit delay. Although the scaling of feature size makes the device smaller and faster, interconnect delay is not scaling down as device delay. Many recent articles reported that interconnect delay can consume as much as 75% of clock cycle in modern designs. Hence, the performance of current designs is mainly determined by interconnect instead of device. In addition, because of the shrinking of device size, the chip area is no longer determined by total cell area, but by the limited routing resources. Extra "white space" is commonly added to provide enough wire tracks to resolve routing congestion. It is typical that more than half of the modern chip is occupied by white space.

Although interconnect is not implemented until the routing stage, its importance makes it necessary to be dealt with in early design stages such as floorplanning and placement. One reason is that floorplanning and placement decides the length and hence the delay of interconnect wires to a large extent. The other is that the white space needs to be allocated appropriately before the routing stage to ensure the routability. Generally speaking, the placement obtained by the design stages before routing determines the solution space for the router to explore. For a bad placement, no matter how good the router is, it is impossible to achieve a good design.

In order to consider the interconnect in early design stages without routing information, many interconnect models are employed to estimate timing and routing congestion for interconnect. To estimate timing, interconnect is modeled by half-perimeter of the bounding box [1] [2] or a star [3] to compute the delay from source to sinks. However, considering the real implementation, multi-pin nets are typically routed as Steiner trees. Hence, both half-perimeter of the bounding box and star-model is far from accurate for interconnect timing estimation. For routing congestion, post-placement congestion estimation methods try to predict the routing congestion for a given placement. In recent years, a number of probabilistic methods for congestion estimation have been proposed [4] [5] [6]. Recently, Westra et al. [7] presented a new technique based on degenerate global routing techniques. All these works proposed generic estimators which aim at predicting the behavior for all routers consistently. However, as pointed out in [8], because routing solutions generated by different routers are very different, it is not possible for an estimator to accurately predict congestion of all routers. Furthermore, even a real global router cannot predict the routing congestion for solutions obtained by another global router. Thus, in both timing and congestion estimation, the interconnect models are far from the real implementation in the routing stage. The interconnect resources required by routing stage are not adequately estimated and reserved during early design stages.

In order to get accurate interconnect information in early design stages, it is desirable to incorporate global routing into them. Global routing allocates the routing demand globally over the chip area. It generates interconnect information very close to the final routing implementation and can be used for accurate estimation of interconnect topology, wirelength, delay, congestion, buffering solution, etc. In addition, if the same global router is used for both early stage interconnect estimation and global routing, the inconsistency between the early design stages and routing can be eliminated.

There are mainly two categories of global routing techniques: rip-up and reroute based techniques, and multicommodity flow based techniques. Many academic routers [9] [10] and the majority of the industry routers employ the rip-up and reroute approach. This kind of techniques are essentially sequential routing methods in which each net is routed in a certain order according to the routing congestion from nets already routed. The multicommodity flow based techniques [11] [12] can handle simultaneous routing of multiple nets as a multicommodity flow problem. The main idea is to model nets as different commodities that flow through the network of routing resource graph. The flow problem is typically solved by linear programming which results in fractional flow. Therefore, a randomized rounding procedure is used to discretize the solution. Albrecht [12] proposed a method to approximate the LP solution with provable error bounds to speed up the computation.

In order to handle large size problems, multilevel routing approaches [17] [18] are proposed to reduce the complexity of the problem. A "V-shaped" recursive coarsening and refinement process is commonly used.

However, due to the high runtime complexity of the traditional global routers, it is impractical to perform global routing repeatedly in early stages. Recently, an extremely fast global router, *FastRoute* [8] was proposed to address the runtime issue. Unlike many global routers which rely on maze routing to resolve the congestion, *FastRoute* focuses on determining good Steiner tree topology and Steiner node locations according to congestion information so that much less maze routing is needed. Experimental results show that *FastRoute* can generate less congested global routing solutions with two orders of magnitude speedup over the state-of-the-art academic global routers *Labyrinth* [9] and *Chi Dispersion* router [10]. And it is even faster than the highly-efficient congestion estimation algorithm *FaDGloR* [7].

In this paper, we propose two major techniques to futher improve

This work was partially supported by the SRC under Task ID 1206 and NSF under grant CCF-0540998.

1-4244-0630-7/07/$20.00 ©2007 IEEE.

FastRoute in solution quality.

- A monotonic routing technique to substitute pattern routing.
- A multi-source multi-sink maze routing technique.

The new router is called *FastRoute* 2.0.

On the same set of benchmarks in [8] [10], *FastRoute* 2.0 achieves much better solution quality than *FastRoute*, *Labyrinth* and *Chi Dispersion* router . The total overflow is reduced by more than an order of magnitude. The runtime is about 73% slower than the extremely fast *FastRoute*, but still 78× and 37× faster than *Labyrinth* and *Chi Dispersion* router.

The remainder of the paper is organized as follows. In Section II, we review the framework and techniques of *FastRoute* global router. In Section III, we present the two major techniques in detail. In Section IV, experimental results of *FastRoute* 2.0 and comparison with three state-of-the-art global routers are shown. Finally, the paper concludes with a summary of results and directions of future work.

II. FastRoute Global Router

In this section, we give an overview of the extremely fast global router, *FastRoute* [8].

Different from traditional global routers, *FastRoute* is a global router aiming at the application in both placement and routing. In placement process, global router may be invoked many times to get the interconnect estimation for intermediate placement. Hence, the runtime is a major concern of the algorithm. As pointed out by many works (e.g, [9]), maze routing is the major contributor of global routing runtime. Therefore, *FastRoute* focuses mainly on the Steiner tree construction to alleviate the burden of maze routing. Because of the good Steiner tree structures obtained, *FastRoute* only runs one round of maze routing and only about 2.15% of 2-pin nets are routed by maze routing. This is the major reason why *FastRoute* can achieve such a significant speedup over other global routers.

FastRoute has three phases:

1) *Congestion map generation:* In this phase, the Steiner trees for all the nets are generated using minimal Steiner tree algorithm *FLUTE* [13]. Then all Steiner trees are broken into 2-pin nets and routed using L-shaped pattern routing. The congestion map is obtained from this rough routing result.
2) *Congestion-driven Steiner tree construction:* In this phase, two major techniques are proposed to construct good Steiner tree structures to reduce the routing congestion. First, a congestion-driven topology generation algorithm generates the Steiner tree topologies to reduce routing congestion according to the congestion map. The algorithm extends the idea of *FLUTE* to handle the congestion by trying to use less wires in the congested region. Second, an edge shifting technique is employed to further reduce the routing congestion after the Steiner tree topology is fixed. It identifies the tree edges that can be shifted without changing the rectilinear wirelength of the tree. By shifting these edges, routing demand can be shifted from congested region to uncongested region so that local congestion can be resolved. Both techniques are applied to every net with more than 4 pins, and the congestion map is updating as each net changes.
3) *Routing of 2-pin nets using pattern routing and maze routing:* In this phase, the Steiner trees obtained from phase 2 are broken into 2-pin nets. Then every 2-pin net is ripped up and rerouted by Z-shaped pattern routing. Finally, the long 2-pin nets over the congested regions is ripped up again and rerouted by maze routing. A cost function based on logistic function [14] is introduced to direct the maze routing to find less congested paths.

FastRoute achieves good global routing solutions with two orders of magnitude faster runtime. The extremely high speed makes it possible to incorporate it directly into the early design stages without much runtime penalty. This could dramatically improve the solution quality because accurate interconnect information becomes available in early stages.

III. New Routing Techniques

The original *FastRoute* focuses on generating good Steiner tree structures to reduce routing congestion in the first two phases. So we follow these two phases of *FastRoute* to generate high-quality Steiner tree structures. However, we demonstrate that the pattern routing and maze routing in the third phase can be improved to obtain better routing solutions. In this work, we propose the following two techniques:

- A monotonic routing to substitute the pattern routing in *FastRoute* flow.
- A multi-source multi-sink maze routing technique which is a more powerful maze routing technique to achieve high-quality routing solutions.

In part A we first discuss the grid graph model used in this work. Then, in B and part C, we will describe the two techniques in detail. Finally, the flow of *FastRoute* 2.0, the new global router based on the two techniques is given in part D.

A. Grid Graph Model

The grid graph model is widely used in global routing [8] [9] [10] [12]. It is also used in our work. In this model, the chip area is partitioned into rectangular regions called global bins and all the pins in a global bin are assumed to be at the center of the bin. Each global bin corresponds to a node in grid graph. The boundaries of global bins are called global edges, which correpond to the edges in grid graph. The capacity of an edge represents the number of routing tracks for the corresponding boundary. These notions are illustrated in Figure 1. The major optimization objective in global routing is to minimize the total overflow on all global edges in the grid graph. The overflow on a global edge e is defined as how much the routing demand d_e exceeds the edge capacity c_e. If $d_e > c_e$, $overflow_e = d_e - c_e$; otherwise $overflow_e = 0$. For each global edge, a cost is associated with it based on its overflow value.

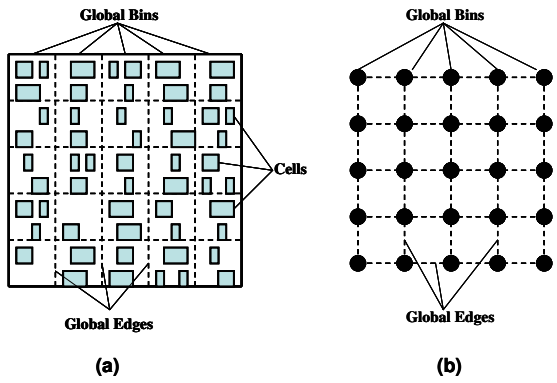

Fig. 1. (a) Global bins. (b) Corresponding grid graph.

B. Monotonic Routing

Pattern routing uses predefined patterns to route 2-pin nets. Usually, the most commonly used are L-shaped (1-bend) or Z-shaped (2-bends) patterns. Because pattern routing limits the pattern of routing

path shapes, it can speed up the global routing process. Therefore, pattern routing is typically employed to route a big portion of nets to save runtime. In *FastRoute*, after every Steiner tree broken into 2-pin nets, Z-shaped pattern routing is used to route each 2-pin net.

Although the pattern routing can speed up the routing process, its quality could be much worse than maze routing. The maze router ensures that the least cost route is found, but pattern routing only considers a small portion of possible routes. For a 2-pin net which spans $m \times n$ grids, L-shaped pattern routing only considers 2 different paths, and Z-shaped pattern routing only considers $m + n$ different paths. Hence, pattern routing fails to find good routing paths to avoid the congestion in many cases. We want to find a trade-off between maze routing and pattern routing so that the quality can be better than pattern routing, but the runtime is close to it.

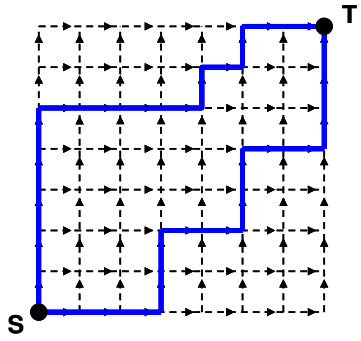

Fig. 2. Monotonic routing paths.

The basic idea is to find the best monotonic routing path for a 2-pin net. Let one pin be the source (S) and the other be sink (T). A monotonic routing path from S to T is a path on the routing grid from S to T which always directs toward T. Figure 2 shows two different monotonic routing paths from S to T. Notice that all monotonic routing paths will not go out of the bounding box of S and T. The total number of monotonic routing paths from one corner to the diagonal corner of a $m \times n$ grids is $\binom{m+n-2}{m-1} = \frac{(m+n-2)!}{(m-1)!(n-1)!}$.

One important property of monotonic routing path is that for every grid point within the bounding box of S and T, only one or two grid points can be its predecessor on any monotonic routing path from S to it. As shown in Figure 3, all the grid points G (empty circles) with the same x-coordinates or y-coordinates as S have only one predecessor G_1, and all other grid points g (solid dots) have two predecessors g_1 and g_2. Without loss of generality, we assume S is at the lower-left corner of the bounding box and T is at the upper-right corner.

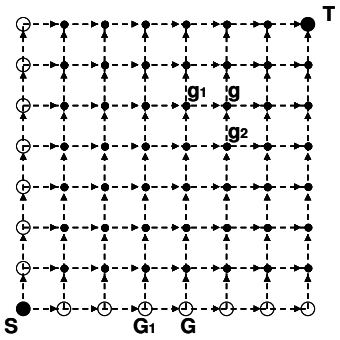

Fig. 3. Monotonic path property.

For the least monotonic path from S to G and S to g, we have the following two lemmas.

Lemma 1: The least cost monotonic routing path from S to G, $P_{SG} = P_{SG1} + (G_1, G)$, where P_{SG1} is the least cost monotonic routing path from S to G_1.

Lemma 2: Let P_1 and P_2 are the least cost monotonic routing paths from S to g_1 and g_2, respectively. The least cost monotonic routing path from S to g, P_{Sg} is one of the following two paths, whichever has less cost. (1) $P_1 + (g_1, g)$, (2) $P_2 + (g_2, g)$.

These two lemmas follow the truth that every monotonic routing path from S to a grid point must be composed of a monotonic routing path from S to its predecessor(s) and the edge between it and its predecessor(s). The two lemmas told us that if the least cost monotonic routing path(s) for the predecessor(s) is found, the least cost monotonic routing path for the current grid point can be found. Hence, we can use dynamic programming to find the least cost monotonic routing path from S to T. The algorithm is shown in Figure 4. Lemma 1 and Lemma 2 ensure the optimality of the algorithm.

Algorithm Monotonic Routing

```
1.  d(S) = 0
2.  for x = 1 to m
3.      G = (x, 0), G1 = (x-1, 0)
4.      d(G) = d(G1) + cost(G, G1), π(G) = G1
5.  for y = 1 to n
6.      G = (0, y), G1 = (0, y-1)
7.      d(G) = d(G1) + cost(G, G1), π(G) = G1
8.  for x = 1 to m
9.      for y = 1 to n
10.         g = (x, y)
11.         g1 = (x-1, y), g2 = (x, y-1)
12.         if d(g1)+cost(g, g1) < d(g2)+cost(g, g2)
13.             d(g) = d(g1)+cost(g, g1), π(g) = g1
14.         else
15.             d(g) = d(g2)+cost(g, g2), π(g) = g2
16. Trace back from T using π to find the least cost monotonic path
```

Fig. 4. Monotonic routing algorithm.

In the algorithm, $S = (0, 0)$, $T = (m, n)$, and $d()$ is the cost of the least cost monotonic path from S to any grid point in the bounding box of S and T. Lines 2-7 finds the least cost for all grid points represented as empty circles in Figure 3. Lines 8-15 finds the least cost for all grid points represented as solid dots in Figure 3, including T. Note that whenever we update d for a grid point, d for its predecessor(s) is already available.

Now we analyze the complexity of the algorithm. Lines 2-4 takes $O(m)$ time, lines 5-7 takes $O(n)$ time, lines 8-15 takes $O(mn)$ time, and line 16 takes $O(m + n)$ time. Hence, the runtime complexity of the algorithm is $O(mn)$, which is the same as Z-shaped pattern routing. In Section IV, experimental results show that monotonic routing is about 2.3× slower than Z-shaped pattern routing.

C. Multi-source Multi-sink Maze Routing

Maze routing is the most popular technique used in global routing. Originally, maze routing algorithm is designed to find the shortest path connecting two pins in the presence of routing blockages. Later, it has been extended to find a path connecting two pins in such a way that it favors a path that passes through less congested area according to some cost function. It is a very powerful technique to find paths avoiding congestion.

However, we notice that the application of maze routing in global routing is to find path between two pins. For multi-pin nets, a typical

way is to break the routing tree into edges (2-pin nets), and route each edge by maze routing. We find that this kind of independent edge-by-edge routing scheme may cause problems and fail to generate good routing solutions for the multi-pin nets. Figure 5 illustrates three different scenarios. The shaded areas denote the congested regions.

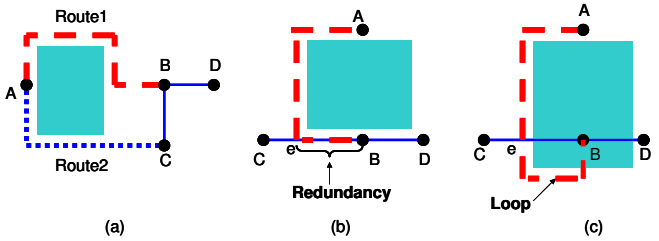

Fig. 5. Maze routing scenarios.

- *Unnecessary detour:* Consider the scenario in Figure 5 (a). The dashed route "Route1" is the maze routing result for edge (A, B). However, if the path does not need to go from A to B, "Route2" is a better choice in terms of cost.
- *Redundant routing:* Consider the scenario in Figure 5 (b). The dashed route is the maze routing result for edge (A, B). However, the (e, B) part on the path is already part of the routing tree, and it is redundant to repeat it.
- *Unintentionally loop:* Consider the scenario in Figure 5 (c). The dashed route is the maze routing result for edge (A, B). A loop is created in the routing tree. It is obvious that this loop is not needed and only the part from A to e is necessary on the path.

As we can see in these three scenarios, unnecessary wires are used in routing the multi-pin nets. This results in using more routing resources than necessary and cause routing congestion. The major defect of this edge-by-edge routing scheme is that the tree information is neglected and every edge is routed independently. When routing an edge in the tree for multi-pin nets, the routing path has to start with one endpoint of the edge and end with the other endpoint. However, this may not be necessary sometimes. It is enough if there is a path created between the two endpoints, no matter it directly goes from one endpoint to the other or uses part of routing tree already there.

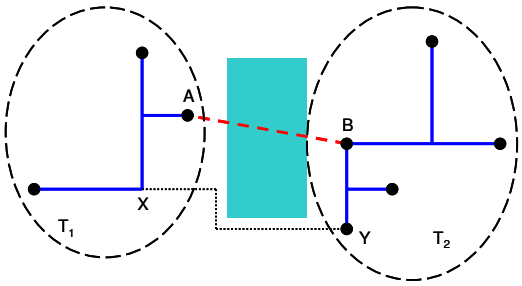

Fig. 6. Multi-source multi-sink maze routing.

Aware of this problem, we propose a multi-source multi-sink maze routing algorithm. The main idea is that the routing tree is respected when we route an edge for a multi-pin net. We do not constrain the two endpoints of the routing path to be the original endpoints of the edge being routed. As illustrated in Figure 6, suppose we are routing an edge (A, B) in the routing tree T for a multi-pin net N. We first remove (A, B) from T and obtain two subtrees T_1 and T_2. (Note that T_1 and T_2 can be just a point.) We treat all the grid points on T_1 as sources, and all the grid points on T_2 as sinks. Then, we apply the multi-source multi-sink maze routing to find the best path connecting T_1 and T_2 to form a tree. In Figure 6, the dotted line from X to Y is the best path to connect T_1 and T_2.

Our multi-source multi-sink maze routing algorithm is shown in Figure 7. In the algorithm, we use the same cost function as in *FastRoute* [8]. $d(g)$ is the distance from T_1 to g, defined as the total cost of all global edges passed by the temporary shortest path from T_1 to g. The algorithm follows the framework of Dijkstra's algorithm [15]. Lines 1-5 initializes the distance d, priority queue Q and destination points. Lines 6-17 is the loop similar to Dijkstra's algorithm. Line 18 just traces back to find the shortest path from T_1 to T_2. Note that Dijkstra's algorithm ensures that when a point u is extracted from Q, $d(u)$ is the shortest distance from T_1 to u. That is why the stopping criterion is when the first destination point is extracted from the priority queue.

Algorithm Multi-source Multi-destination Maze Routing

```
1.  d(g) = ∞ for all grid points g
2.  Find subtree T1 (contains A) and T2 (contains B) after breaking (A, B)
3.  Set d(u) = 0 and π(u) = nil, for all grid points u on T1
4.  Set up a priority queue Q with all grid points on T1
5.  Mark all grid points on T2 as destination point
6.  u ← Extract-Min(Q)
7.  While u is not destination point
8.      do
9.          for each neighbor grid points v of u
10.         do
11.             if d(v) > d(u) + cost(u, v)
12.                 then  d(v) = d(u) + cost(u, v)
13.                       π(v) = u
14.                       if v is in Q
15.                           then  update Q
16.                           else  insert v into Q
17.         u ← Extract-Min(Q)
18. Trace back from u using π to find the shortest path from T1 to T2
```

Fig. 7. Multi-source multi-sink maze routing algorithm.

Our algorithm finds the least cost routing path from T_1 to T_2. Theorem 2 gives the optimality of the algorithm.

Theorem 2 The path found by multi-source multi-sink maze routing algorithm is the least cost routing path from T_1 to T_2.

Due to the page limit, we only give the sketch of the proof.

Proof: First of all, note that the cost function cost(u, v) is a positive function in our problem. In line 3, $d(u) = 0$ for all the grid points on T_1. Hence, we can assume a supersource which replaces all the grid points on T_1, and all grid points adjacent to T_1 are its neighbor. Similarly, we can assume a supersink which replaces all the grid points on T_2, and all adjacent grid points to T_2. Then the problem is transformed to a single-source, single-sink shortest path problem. The optimality follows the optimality of Dijkstra's algorithm.

The only thing left is to prove the stopping criterion is correct. Recall that we stop when a destination point on T_2 is extracted from Q. Assume u is the first destination point extracted from Q. For the purpose of contradiction, let w be the destination point which is on the shortest path from T_1 to T_2. Hence, we have $d(w) < d(u)$. However, when we extract u from Q, w is still in Q, which means $d(w) \geq d(u)$. Because the cost function is positive, $d(w)$ will never decrease in later updating. Therefore, we obtain a contradiction that $d(w) \geq d(u)$. ■

Now we analyze the complexity of the algorithm. Assume there are V grid points in the search region. Lines 1-5 takes time $O(V)$. Each Extract-Min operation on the priority queue Q takes time $O(lgV)$. There are at most V iterations for the while loop. For each

u, there are at most 4 neighbors adjacent to it. The insertion and updating of Q takes time $O(lgV)$. The total complexity is therefore $O(VlgV)$.

We apply this multi-source multi-sink maze routing algorithm on the tree edges of multi-pin nets. The runtime of maze routing algorithm is highly related to the size of the search region. In order to speed up the algorithm, we do not always search the whole grid graph to find the least cost path. Instead, we expand each boundary of the bounding box by a certain amount, say w rows and h columns, and use this enlarged region as the search region for the maze routing algorithm. By using this kind of search region, the runtime can be reduced significantly but the solution quality is close to optimal. In our implementation, the enlarge value is 10 for all four boundaries in the first maze routing round. If more rounds are needed, the enlarge value is increased by 10 every round.

We want to point out one issue for the multi-source multi-sink maze routing technique. It can totally change the routing tree structure because the endpoints of new routing path do not need to be the endpoints of the edge being routed. For example, in Figure 8, the Steiner tree structure is changed from (a) to (b) because of the new routing of edge (A, B). Hence, we need to update the Steiner tree structure accordingly after routing each edge by multi-source multi-sink maze routing.

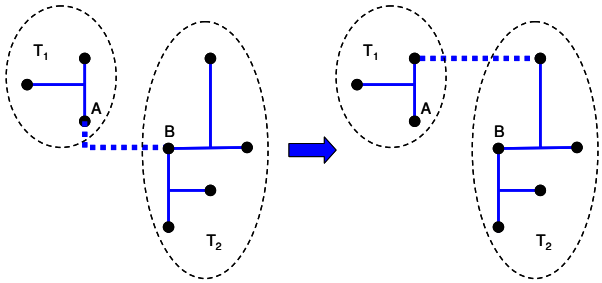

Fig. 8. Steiner tree topology changed by maze routing.

D. Flow of FastRoute 2.0

In this part, we give the flow of the new router, *FastRoute* 2.0. In general, *FastRoute* 2.0 has the flow similar to *FastRoute*. First, the congestion map is generated. Second, Steiner tree structures are constructed according to the congestion map. Finally, monotonic routing and multi-source multi-sink maze routing are applied to route the tree edges in Steiner routing trees.

The first two phases are the same as *FastRoute*. In the final phase, we first apply the monotonic routing to every edge in every routing tree. Then we run one round of multi-source multi-sink maze routing. However, in this maze routing round, we are not routing every edge by maze routing. Instead, only the edges longer than a threshold and across congested areas will be routed by maze route, and other edges are routed by monotonic routing. The intuition is to avoid routing short nets and long nets not passing congested area by maze routing. Otherwise, unnecessary detours may be created to use more wirelength and cause routing congestion. Of course, another important reason is to cut down the runtime of maze routing. If there is still a lot of overflow, we will run more rounds of maze routing.

IV. Experimental Results

In this section, we present the experimental results. All experiments were performed on a Linux workstation with Intel Pentium 4 3.0 GHz CPU and 2GB memory.

TABLE I
BENCHMARK STATISTICS

	Grids	# Nets	# Routed Nets	Max Deg	Avg Deg
ibm01	64x64	11.5k	9.1k	37	3.8
ibm02	80x64	18.4k	14.3k	126	4.4
ibm03	80x64	21.6k	15.3k	49	3.6
ibm04	96x64	26.2k	19.7k	41	3.4
ibm06	128x64	33.4k	25.8k	34	3.8
ibm07	192x64	44.4k	34.4k	22	3.8
ibm08	192x64	47.9k	35.2k	65	4.3
ibm09	256x64	50.4k	39.6k	38	3.8
ibm10	256x64	64.2k	49.5k	32	4.2

First, we compare *FastRoute* 2.0 with three state-of-the-art academic global routers: *FastRoute* [8], *Labyrinth* [9] and *Chi Dispersion* router [10]. We use the same benchmarks as in [10] provided by the authors of [9]. Statistics of the benchmark circuits are shown in Table I. Because several pins in a net may fall in the same grid, the number of routed nets is less than the total number of nets. For *Labyrinth*, 70% of the shortest connections are routed by pattern routing, which is the same as in [10]. We measure wirelength and total overflow in the same manner as [8] and [10]. The results are summarized in Table II. *FastRoute* 2.0 can achieve 0 overflow for 6 circuits out of the total 9 circuits, and the total overflow is reduced by more than an order of magnitude compared to the other three routers. The wirelength of *FastRoute* 2.0 is also the least among all the routers. At the same time, *FastRoute* 2.0 is 73% slower than *FastRoute*, but 78× and 37× faster than *Labyrinth* and *Chi Dispersion* router, respectively. Because we cannot find a version to duplicate the results in [10], the runtime of *Chi Dispersion* router is scaled from the runtime in [10] based on the information from Standard Performance Evaluation Corporation (SPEC) [16]. In [10], it was claimed that runtime of *Chi Dispersion* router is roughly 2× faster than *Labyrinth*, which coincides with the scaled runtime we obtained. We also get a new version of *Chi Dispersion* router from the authors of [10], the total overflow on the same set of benchmark is 804, but the total runtime is about 65× slower than *FastRoute* 2.0, which is close to the runtime of *Labyrinth*. In [8], a beaver mode of *FastRoute* with more maze routing is also reported. The total overflow of beaver mode is 512 and it is 2.2× slower than *FastRoute* default mode, which is worse than *FastRoute* 2.0 in both total overflow and runtime. This indicates that just applying more maze routing in *FastRoute* cannot achieve the high-quality results of *FastRoute* 2.0. Recently, a high-quality global router, *BoxRouter* [19] was proposed. Because of page limit, we are not able to include all the comparison results with *BoxRouter*. On the same set of benchmark, the total congestion for *BoxRouter* is 497, compared to 98 for *FastRoute* 2.0. And the runtime of *BoxRouter* is about 30× slower than *FastRoute* 2.0.

Second, we investigate the effect of monotonic routing technique. In order to show the effect of monotonic routing, we set up 4 different flows for phase 3.

- Only Z-shaped pattern routing
- Only Monotonic routing
- Z-shaped pattern routing + Maze routing
- Monotonic routing + Maze routing

Table III shows the comparison results between the first and second flow, as well as between the third and fourth flow. It is clear that monotonic routing can generate less congested solutions before and after the maze routing. And we also measure the runtime for one full round of monotonic routing and one full round Z-shaped pattern routing. The previous one is about 2.3× slower than the latter. But one point we want to mention that sometimes pattern routing may be preferred because it may generate less vias. When there is strict

TABLE II
COMPARION OF FASTROUTE 2.0, FASTROUTE, LABYRINTH AND CHI DISPERSION ROUTER

	FastRoute 2.0			FastRoute			Labyrinth Predictable router			*Chi* Dispersion router		
	Ovflow	Wirelen	Time(s)	Ovflow	Wirelen	Time(s)	Ovflow	Wirelen	Time(s)	Ovflow	Wirelen	Time(s)*
ibm01	31	68489	0.72	250	67128	0.21	242	76228	16.99	189	66005	8.63
ibm02	0	178868	0.93	39	179995	0.56	214	202235	26.53	64	178892	26.27
ibm03	0	150393	0.60	1	151023	0.43	117	191500	37.92	10	152392	24.71
ibm04	64	175037	1.88	567	172593	0.50	786	198181	80.95	465	173241	32.94
ibm06	0	284935	1.36	33	285882	0.91	130	339379	72.06	35	289276	53.33
ibm07	0	375185	1.60	18	376835	1.05	407	450855	168.41	309	378994	79.61
ibm08	0	411703	2.36	58	412915	1.16	352	466556	154.82	74	415285	72.94
ibm09	3	424949	1.92	28	426471	1.39	310	481841	229.59	52	427556	86.67
ibm10	0	595622	2.79	18	599433	1.98	288	680113	296.70	73	599937	139.61
Total	98	2665181	14.16	1012	2672275	8.19	2846	3086888	1083.97	1271	2681578	524.71
Norm**	**1**	**1**	**1**	**10.327**	**1.003**	**0.578**	**29.041**	**1.158**	**77.552**	**12.969**	**1.006**	**37.056**

(*) Scaled runtime on our machine. (**) Normalized to FastRoute 2.0 results.

constraint on vias, pattern routing may be a good choice.

TABLE III
OVERFLOW VALUES OF DIFFERENT FLOWS

	Z	Monotonic	Z + maze	Monotonic + maze
ibm01	1435	1280	40	31
ibm02	2711	2569	0	0
ibm03	260	145	0	0
ibm04	1950	1794	112	64
ibm06	1682	1444	0	0
ibm07	1020	853	0	0
ibm08	963	735	1	0
ibm09	1065	626	21	3
ibm10	1834	1532	2	0
Total	12920	10978	176	98

Third, we report the total number of tree edges, the percentage of tree edges been maze routed, and the percentage of tree edges whose endpoints are changed during multi-source multi-sink maze routing. From Table IV, we can see that only 2.34% of edges are maze routed, which is the main reason why the algorithm is very fast. Also, the results show that a significant portion (1 out of 3.5) of the edges been maze routed have their endpoints changed during the multi-source multi-sink maze routing. This indicates the effectiveness of not constraining the endpoints of the routing path for the edges.

TABLE IV
MAZE ROUTING STATISTICS

	Total # of tree edges	Edges being maze routed (%)	Edges w/ endpoints changed (%)
ibm01	28116	3.24%	1.12%
ibm02	55361	4.00%	1.25%
ibm03	45582	1.79%	0.45%
ibm04	53308	4.04%	0.88%
ibm06	82283	2.43%	0.62%
ibm07	109175	1.89%	0.44%
ibm08	133222	1.13%	0.30%
ibm09	128185	1.25%	0.45%
ibm10	181432	1.25%	0.47%
Avg		**2.34%**	**0.66%**

V. CONCLUSIONS

In this paper, we develop a high-quality and very efficient global router - *FastRoute* 2.0. It can generates routing solutions with an order of magnitude less overflow. Its runtime is 73% slower than *FastRoute* but still much faster than *Labyrinth* and *Chi Dispersion*.

Our future work will focus on two aspects. First, we will incorporate our technique into the multi-level framework to handle very large routing problems. Second, we will integrate *FastRoute* 2.0 into placement framework to develop placement algorithms that generate better solutions in terms of timing, routability, etc.

REFERENCES

[1] H. Eisenmann and F. M. Johannes. Generic global placement and floorplanning. In *Proc. ACM/IEEE Design Automation Conf.*, pp. 269-274, 1998.

[2] X. Yang, B.-K. Choi, and M. Sarrafzadeh. Timing-driven placement using design hierarchy guided constraint generation. In *Proc. IEEE/ACM Intl. Conf. Computer-Aided Design*, pp. 177-184, 2002.

[3] G. Stem, B. M. Riess, B. Rohfleisch and F. M. Johannes. Timing driven placement in interaction with netlist transformations. *Proc. Intl. Symp. on Physical Design*, pp. 36-41, 1997.

[4] J. Lou, S. Krishnamoorthy, and H. Sheng. Estimating routing congestion using probabilistic analysis. In *Proc. Intl. Symp. on Physical Design*, pp. 112-117, 2001.

[5] J. Westra, C. Bartels, and P. Groeneveld. Probabilistic congestion prediction. In *Proc. Intl. Symp. on Physical Design*, pp. 204-209, 2004.

[6] A. B. Kahng and X. Xu. Accurate pseudo-constructive wirelength and congestion estimation. In *Proc. Intl. Workshop on System-Level Interconnect Prediction(SLIP)*, pp. 61-68, 2003.

[7] J. Westra and P. Groeneveld. Is probabilistic congestion estimation worthwhile? In *Proc. Intl. Workshop on System-Level Interconnect Prediction(SLIP)*, pp. 99-106, 2005.

[8] M. Pan and C. Chu. FastRoute: A step to integrate global routing into placement. To appear. *IEEE/ACM Intl. Conf. Computer-Aided Design*, 2006.

[9] R. Kastner, E. Bozogzadeh, and M. Sarrafzadeh. Predictable routing. In *Proc. IEEE/ACM Intl. Conf. Computer-Aided Design*, pp. 110-113, 2000.

[10] R. T. Hadsell and P. H. Madden. Improved global routing through congestion estimation. In *Proc. ACM/IEEE Design Automation Conf.*, pp. 28-31, 2003.

[11] R. Carden, J. Li, and C. K. Cheng. A global router with a theoretical bound on the optimal solution. In *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, 15(2):208-216, 1996.

[12] C.Albrecht. Global routing by new approximation algorithms for multicommodity flow. In *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, 20(5):622-631, 2001.

[13] C. Chu, Y. Wong. Fast and Accurate Rectilinear Steiner Minimal Tree Algorithm for VLSI Design. In *Proc. Intl. Symp. on Physical Design*, pages 28-35, 2005.

[14] D. von Seggern. CRC Standard Curves and Surfaces. Boca Raton, FL: CRC Press, 1993.

[15] E. Dijkstra. A note on two problems in connexion with graphs. In *Numerische Mathematik, vol. 1*, pp. 269-271, 1959.

[16] http://www.spec.org/

[17] J. Cong, J. Fang and Y. Zhang. Multilevel Approach to Full-Chip Gridless Routing. In *Proc. IEEE/ACM Intl. Conf. on Computer-Aided Design*, pp. 396-403, 2001.

[18] T. Y. Ho, Y. W. Chang, S. J. Chen, and D. T. Lee. A fast crosstalk and performance-driven multilevel routing system. In *Proc. IEEE/ACM Intl. Conf. on Computer-Aided Design*, pp. 382-387, 2003.

[19] M. Cho and D. Z. Pan. BoxRouter: a new global router based on box expansion and progressive ILP. In *Proc. ACM/IEEE Design Automation Conf.*, pp. 373-378, 2006.