FastRoute3.0: A Fast and High Quality Global Router Based on Virtual Capacity

Yanheng Zhang, Yue Xu and Chris Chu
Department of Electrical and Computer Engineering
Iowa State University, Ames, IA 50011
Email: {zyh, yuexu, cnchu}@iastate.edu

***Abstract*— As an easily implemented approach, ripup and reroute has been employed by most of today's global routers, which iteratively applies maze routing to refine solution quality. But traditional maze routing is susceptible to get stuck at local optimal results. In this work, we will present a fast and high quality global router FastRoute3.0, with the new technique named virtual capacity. Virtual capacity is proposed to guide the global router at maze routing stage to achieve higher quality results in terms of overflow and runtime. During maze routing stage, virtual capacity works as a substitute for the real edge capacity in calculating the maze routing cost. There are two sub techniques included: (1) virtual capacity initialization, (2) virtual capacity update. Before the maze routing stage, FastRoute3.0 initializes the virtual capacity by subtracting the predicted overflow generated by adaptive congestion estimation (*ACE*) from the real edge capacity. And in the following maze routing iterations, we further reduce the virtual capacity by the amount of existing overflow(edge usage minus real edge capacity) for the edges that are still congested. To avoid excessive "pushing-away" of routing wires, the virtual capacity is increased by a fixed percentage of the existing overflow if edge usage is smaller than real edge capacity.**

Experimental results show that FastRoute3.0 is highly proficient dealing with ISPD98, ISPD07 and ISPD08 benchmark suites. The results outperform published ripup and reroute based academic global routers in both routability and runtime. In particular, (1) FastRoute3.0 completes routing all the ISPD98 benchmarks. (2) For ISPD07 and ISPD08 global routing contest benchmarks, it generates 12 out of 16 congestion free solutions. (3) The total runtime is enhanced greatly.

I. Introduction

As the feature size of modern VLSI design continues to shrink, the ascending circuit density poses greater challenges for modern chip routers. Modern designs are liable to congestion problems because of the increasingly concentrated routing demands. Besides, due to the rapidly growing problem size, the runtime required for completing a routing task is much longer than before.

There are two major stages for handling the routability problem: placement stage and routing stage. Placement determines the instance and pin locations and hence the degree of difficulty for the routing problem that follows. Min and Chu [6] pointed out that to guide the placer to produce a routable placement, the routing estimation during placement should be consistent with the actual routing in the routing stage. It is desirable to have a fast router that can be repeatedly called right after the placement step for a quick estimation. The estimating router will be identical with the one used in the routing stage. From this sense, modern global routers need better take both runtime and routability issues into consideration.

In VLSI routing, global routing is an essential phase of the whole routing scheme, which determines routing usage based on an abstracted grid graph. Many global routing techniques have been proposed since the 1960s. The most popular global routing approach is iterative ripup and reroute based. The approach first breaks each multi-pin net into a set of two-pin nets. Then the two-pin nets are routed sequentially based on a predetermined order. The routing solution is iteratively refined by rip-up and reroute until reaching acceptable quality. However, the heuristic nature of such an approach is prone to getting stuck at local optimal solutions.

There have been several methods proposed to boost the quality of iterative ripup and reroute approach. Kastner et al. [2] proposed a pattern based routing scheme. Hadsell and Madden [3] proposed to guide the routing by amplifying the congestion map with a new congestion cost function. BoxRouter [5] proposed a hybrid approach with the application of ILP to simultaneously handle multiple nets and achieved better routability. FastRoute [6] achieved very fast runtime by exploring congestion driven RSMT to avoid the extensive usage of maze routing. FastRoute2.0 [7] improved the solution quality over FastRoute by introducing monotonic routing and multi source multi sink maze routing. Recently, Archer [9], BoxRouter2.0 [8], FGR [10], and NTHU-R [12] are presented. All these four techniques employ the negotiation-based maze routing that was introduced by PathFinder [1]. Negotiation-based maze routing increases the maze routing cost for the edges that are consistently congested.

In this paper, we propose the virtual capacity technique which is a systematic way of tackling the congestion problem. As implied by the name, virtual capacity idea tries to guide ripup and reroute global router in the maze routing stage by the "virtual" capacities, instead of the real ones. Given a global routing solution, consider any congested routing edge e. Assume edge e has capacity c_e, routing demand u_e and overflow $o_e (= u_e - c_e > 0)$. The basic idea of virtual capacity is to reduce the capacity of e by o_e units (i.e., set the virtual capacity to $c_e - o_e$) and then run another round of global routing. Because of the reduction in capacity, edge e becomes more expensive to use and hence some of its routing demand will be pushed away. In the ideal situation, exactly o_e units of routing demand will be pushed away in order to bring the congestion back to the level of the previous round. Thus, the new routing demand will be $u_e - o_e = c_e$, i.e., the same as the original capacity. In order words, edge e will not be congested in the second round of global routing. In reality, less than o_e units of routing demand will get pushed away because other edges may not be willing to absorb the pushed routing demand. Nevertheless, virtual capacity is a systematic way to effectively reduce the overflow.

In FastRoute3.0, the virtual capacity is initialized by reducing the real capacity with the amount of estimated overflow generated by adaptive congestion estimation(*ACE*). During the following maze routing process, it is further reduced by the amount of existing overflow if the edge is still congested. Otherwise it is increased by timing a factor to prevent excessive subtraction.

We develop FastRoute3.0 by integrating the new techniques into FastRoute2.0 [7], a fast rip-up and reroute based global router. Compared with the other published academic global routers, FastRoute 3.0 achieves much better results in both routability and runtime. In particular, it completes routing all the ISPD98 benchmarks. For the ISPD07 and ISPD08 global routing contest benchmarks, it successfully generates 12 out of 16 congestion free solutions in very short runtime.

978-1-4244-2820-5/08/$25.00 ©2008 IEEE

The next few sections are organized as follows: Section 2 describes the framework of FastRoute3.0. Section 3 presents the virtual capacity idea. Section 4 introduces two techniques that are effective in via reduction and convergence speedup. Section 5 discusses experimental results and comparisons. Conclusion is made in the section 6.

II. PRELIMINARIES

A. Grid Graph Model

As is illustrated in figure 1, the whole routing region is partitioned into a number of global bins. Each global bin is represented by one node and each common boundary is represented by one edge in the grid graph. The edge is called global edge with the capacity of c_e. The overflow is defined as how much is usage u_e over the c_e. If u_e is smaller than c_e, $o_e = 0$, otherwise $o_e = u_e - c_e$.

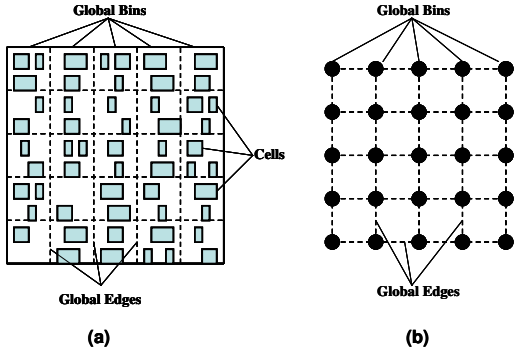

Fig. 1. Global bins and corresponding global routing grid graph.

B. Overview of FastRoute3.0

The flow of FastRoute3.0 is illustrated in figure 2. There are six major steps in the flow. Step 1, 3, 4 are techniques that we borrow from FastRoute2.0. The step 1 is multi-pin nets decomposition. We utilize FLUTE2.5 [4] to generate the congestion driven RSMT. Then the RSMT of all the multi-pin nets are decomposed into a group of two-pin nets. The step 2 is the virtual capacity initialization technique which will be discussed in section 3. Step 3 is pattern routing. Normally L routing and Z routing are applied in this step. Step 4 is multi source multi sink maze routing. The maze routing cost is calculated by adaptive maze function based on virtual capacity. And step 5 is the virtual capacity update which is performed at the end of each maze routing iteration. The virtual capacity value will be either increased or decreased depends on current edge usage compared with original edge capacity. Step 4 and step 5 will be iteratively applied until the total overflow gets stuck. Step 6 performs layer assignment after the program runs out of the iterative ripup and reroute loop.

III. VIRTUAL CAPACITY TECHNIQUE

In this section, we will present the virtual capacity technique. Congestion reduction is key metric to evaluate a global router. Recent published academic global routers including [8], [10], [11] and [12] employ negotiation based maze routing technique, which increments the maze routing cost for edges that are consistently congested. In FastRoute3.0, we propose virtual capacity, an alternative but systematic method to handle congestion problem. Virtual capacity technique consists of two ideas. Section III.A discusses the virtual capacity initialization. Section III.B describes the criteria how virtual capacity is updated.

```
FastRoute3.0 Framework
begin
   Step1 : Congestion Driven Multi-Nets Decomposition
   Step2 : Virtual Capacity Initialization
   Step3 : Pattern Routing (L and Z Routing)
   while( total overflow not gets stuck)
      Step4 : Multi Sink Multi Source Maze Routing
      Step5 : Virtual Capacity Update
   end while
   Step6 : Layer Assignment
end
```

Fig. 2. FastRoute3.0 framework

A. Virtual Capacity Initialization by ACE

Virtual Capacity is initialized by subtracting the estimated overflow from the real edge capacity. In FastRoute3.0 we use adaptive congestion estimation(*ACE*) technique to predict the overflow. The idea is to assign net usage to proper routing edge and calculate the estimated overflow accordingly. Since before the maze routing, each decomposed two-pin net is routed without detour inside the bounding box. And the task of a global router, in general, is to distribute routing demand. The estimation is therefore based on the following two assumptions: (1) Estimating region of each two-pin net is confined within the bounding box. (2) Fractional usage assignment is allowed. The first assumption suggests that we only consider the routing edges inside the bounding box. The second assumption permits breaking the integer usage into fractional values. It keeps in accordance with the objective to evenly distribute the net usage, because fractional values offer more freedom in filling routing demands.

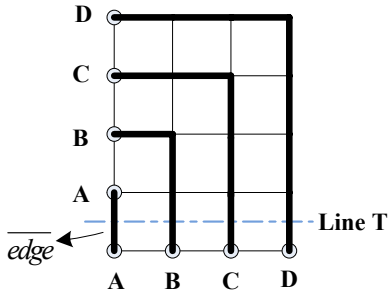

Fig. 3. Probabilistic estimation.

For the runtime consideration, the estimation needs be kept simple and effective. The simplest way of predicting congestion is probabilistic based. For instance, figure 3 illustrates a routing example with four two-pin nets, A, B, C and D. Here we suppose: (1) the edge capacity is 1; (2) estimation goes inside each bounding box; (3) each two-pin net has equivalent probability to pass through the edges along the same row or column. In this case, net D has $1/4$ of probability of passing each of the vertical edge along line T. The same rule applies, the total probabilistic usage of the leftmost vertical edge, here we denotes as $\overline{edge}$, will become: $(1 + 1/2 + 1/3 + 1/4)$. Likewise, for the case with N such nets, the probabilistic usage of $\overline{edge}$ will be: $U(\overline{edge}) = (1 + 1/2 + 1/3 + \cdots 1/N)$. The function is also called harmonic function which diverges when N approximates infinity. In other word, the function will generate a large estimated usage for $\overline{edge}$. However, as depicted in figure 3, the routing case, regardless of the value of N, is entirely routable without detour.

Hence the desired estimated usage of $\overline{edge}$ is 1. It fully shows the deficiency of the traditional probabilistic usage assignment technique.

FastRoute3.0 utilizes *ACE* instead to perform the usage assignment. The notation of problem formulation is shown in table 1. Each two-pin net has the usage of 1, and the objective is to assign the usage to global routing edges more evenly. As the example discussed above, the problem is too much usage is piled on some edge, which could be assigned elsewhere. Also, since the usage assignment is applied in a sequential order, the permutation method prominently affects the accuracy. Therefore in sum, there are two sub problems involved: (1) the usage assignment for single two-pin net; (2) the sequential assignment ordering of a group of nets.

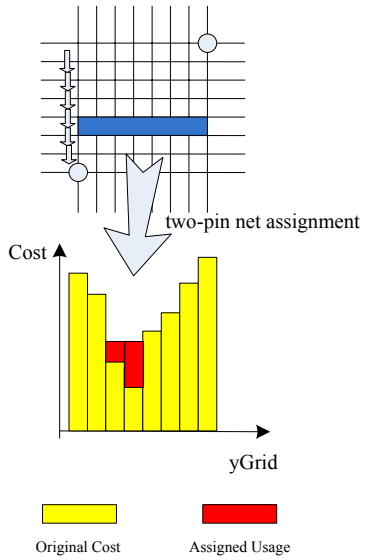

Fig. 4. two-pin net usage assignment(vertical case).

TABLE I
ACE USAGE ASSIGNMENT NOTATION

N	number of two-pin nets
$BBox_k$	bounding box of net_k
r_k	number of rows inside $BBox_k$
c_k	number of columns inside $BBox_k$
$left_k$	left coordinate of $BBox_k$
$right_k$	right coordinate of $BBox_k$
top_k	top coordinate of $BBox_k$
$bottom_k$	bottom coordinate of $BBox_k$
$c_{i,j}^{V/H}$	capacity of the $edge_{i,j}^{V/H}$
$p_{i,j}^{V/H}$	current assigned usage of $edge_{i,j}^{V/H}$

1) Two-pin net assignment: Consider the usage assignment of one single two-pin net, the usage ready to be assigned within the bounding box is 1. Without loss of generality, here we discuss the assignment for vertical edges. The same criteria will be applied for horizontal edges. The usage assignment algorithm for vertical edges is shown in figure 5. Each row is processed independently. Inside one row, edges are sorted in a decreasing order according to the value of $cost_{i,j}^V$, which is equal to $p_{i,j}^V + m_i^V - c_{i,j}^V$. m_i^V is the value of maximum edge capacity of row i. The algorithm compares the average potential assigned usage with largest current assigned usage. It iteratively excludes the edge with largest current assigned usage until an even assignment is possible. The time complexity required for processing single two-pin net net_k is $O(r_k c_k \cdot log(c_k))$. Figure 4 illustrates the assignment process.

Algorithm ACE two-pin net assignment vertical(net_k)

begin
for $(i = top_k \cdots bottom_k + 1)$
  $m_i^V = max(c_{i,j}^V), j \in [left_k, right_k]$
  $\Delta = right_k - left_k + 1$
  for $(j = left_k \cdots right_k)$
    $cost_{i,j}^V = p_{i,j}^V + m_i^V - c_{i,j}^V$
  $C_{sum} = \sum cost_{i,j}^V \ (j \in [left_k, right_k])$
  Sort $cost_{i,j}^V (j \in [left_k, right_k])$ by decreasing order
  Copy sorted edge index into queue Q
  for $(t = 1 \cdots \Delta)$
    if $\frac{1+C_{sum}}{\Delta - t + 1} > cost_{i,Q(t)}^V$
      for $(n = t \cdots \Delta)$
      $p_{i,Q(n)}^V = \frac{1+C_{sum}}{\Delta - t + 1} - m_i^V + c_{i,Q(n)}^V$
      break out of the second for loop
    else
      $C_{sum} = C_{sum} - cost_{i,Q(t)}^V$
  end for
end for
end

Fig. 5. The ACE two-pin net assignment algorithm for vertical edges

2) Net processing order: *ACE* chooses to process smaller span nets with higher priority. The net span represents width of bounding box in vertical edge assignment and likewise height of bounding box in horizontal edge assignment. In experiment we discover that nets with larger spans offer more choices to distribute the net usage. Therefore we permutate the net by net span and perform usage assignment for smaller span nets first. Figure 6 shows the detail of the whole *ACE* algorithm.

Algorithm ACE usage assignment()

begin
$p_{i,j}^V = 0 \ \ \forall i, j$
$p_{i,j}^H = 0 \ \ \forall i, j$

Sort two-pin nets by BBox width with increasing order
Copy sorted nets into queue Q^V
for $(t = 1 \cdots N)$
  ACE two-pin net assignment vertical($Q^V(t)$)
end for

Sort two-pin nets by BBox height with increasing order
Copy sorted nets into queue Q^H
for $(t = 1 \cdots N)$
  ACE two-pin net assignment horizontal($Q^H(t)$)
end for
end

Fig. 6. The ACE usage assignment algorithm

Now we apply *ACE* to solve the routing example in figure 3. Consider vertical edges along the line T as before. Due to the permutation, the net processing order becomes A→B→C→D. After assigning net A, current assigned usage becomes(1,0,0,0). And we will get (1,1,0,0) after assigning net B. As it goes on, the final assigned usage will be (1,1,1,1). So the estimation won't report any potential congestion, which matches exactly with the analysis.

After the estimation, virtual capacity will be initialized by

equation 1.

$$vc_e = rc_e - (max(0, p_e - rc_e)) \quad \forall e \tag{1}$$

In the equation, rc denotes real edge capacity, p is the final assigned usage obtained by *ACE*. The new capacity after subtraction is named virtual capacity, which is vc in abbreviation.

Now we analyze the time complexity of *ACE* technique. The ordering of N two-pin nets takes $O(N \cdot log(N))$. For each net, the worst case time complexity is $O(G^2 log(G))$, G is the maximum number of horizontal and vertical grids. Hence, in general, the overall worst case time complexity is $O(N \cdot log(N) + NG^2 \cdot log(G))$. But the bounding box of a two-pin net is generally small, therefore on average, *ACE* accounts for nearly 20% of total runtime.

B. Virtual Capacity Update

After the virtual capacity initialization, FastRoute3.0 substitutes virtual capacity for the real edge capacity to guide maze routing. But as the ripup and reroute proceeds, especially after several iterations, the initial virtual capacity value becomes less effective. Sometimes it even misleads the router. One reason for causing this phenomena is that the congestion estimation is performed within the bounding box. However, that assumption is not supported by maze routing, which is very likely to generate a lot of detour during rip up and reroute iterations. Therefore, to fix this inconsistency, the virtual capacity needs be updated dynamically. In FastRoute3.0, it is updated at the end of each maze routing iteration.

The update method is presented in equation 2 and 3. Existing overflow o_e is calculated as the difference between edge usage u_e and real edge capacity rc_e. Virtual capacity will be monotonically decreased for the edges that are consistently congested.

However, we could notice that the virtual capacity reduction procedure is irreversible and the capacity is continually lost. So it's highly possible that more and more extra wirelenth will be created by the edges with very small virtual capacity, even though some edges are not congested at all. As a result, we apply virtual capacity increase if the edge usage is below real edge capacity(o_e is less than 0). In equation 3 the augmenting factor F is set to be 0.85 by experiment.

$$o_e = u_e - rc_e \quad \forall e \tag{2}$$

$$vc_e = \begin{cases} vc_e - o_e & \text{if} \quad o_e \geq 0 \\ vc_e - F \times o_e & \text{if} \quad o_e < 0 \end{cases} \tag{3}$$

Virtual capacity technique is effective in overflow reduction. Table 2 compares the two modes(with and without virtual capacity) on seven routable ISPD07 global routing benchmarks.

TABLE II
COMPARISON OF DIFFERENT MODES ON 7 ROUTABLE 3D VERSION ISPD07 BENCHMARKS

name	mode1			mode2		
	wlen (e5)	cpu (sec)	OF	wlen (e5)	cpu (sec)	OF
adaptec1	55.1	302	0	/	1800	42
adaptec2	53.6	38	0	/	1800	312
adaptec3	133	249	0	132	639	0
adaptec4	122	54	0	122	77	0
adaptec5	161	682	0	/	1800	436
newblue1	48.2	316	0	/	1800	1348
newblue2	76.3	24	0	76.2	36	0

In the table, mode 1 utilizes virtual capacity in maze routing and mode 2 is traditional maze routing, which switches off virtual capacity technique. Maze routing with virtual capacity apparently shows much better performance in terms of congestion reduction and runtime. Traditional maze routing could only finish routing 3 benchmarks. Note that the runtime is limited to be within 30 minutes.

IV. Quality Improvement Techniques

In this section, we will present another two simple but effective techniques: (1) via aware maze routing; (2) adaptive maze function.

A. Via aware maze routing

In FastRoute3.0, almost about half of the vias are generated by routing bends. To suppress the number of vias, via cost is incorporated into the maze routing cost function. During the dijkstra expansion, we record the predecessor of current grid position. If the new expansion causes any routing bends, via cost is added to the maze routing cost.

$$cost_e = cost_e + viacost \tag{4}$$

TABLE III
COMPARISON OF VIA AWARE MAZE ROUTING AND VIA IGNORED MAZE ROUTING ON 7 ROUTABLE 3D VERSION ISPD07 BENCHMARKS

name	via ignored maze routing			via aware maze routing		
	#via	cpu (sec)	OF	#via	cpu (sec)	OF
adaptec1	2007K	293	0	1843K	302	0
adaptec2	2033K	35	0	1989K	38	0
adaptec3	3848K	238	0	3600K	249	0
adaptec4	3306K	51	0	3287K	54	0
adaptec5	5667K	672	0	5489K	682	0
newblue1	2425K	312	0	2314K	316	0
newblue2	3009K	21	0	2992K	24	0

In table 3 we compare the via aware maze routing and via ignored maze routing on seven routable ISPD07 3D benchmarks. We could observe that the extra via cost could effectively remove over 3% 3D vias with 2% increase of runtime.

B. Adaptive Maze Function

The adaptive maze routing cost function is presented in equation 5. In the function. k is the coefficient controlling the function curve slope when u_e is below c_e. k is adaptively adjusted in different maze routing phases. In the initial phase, the k is set small to preserve good wirelegnth. Normally in the first few iterations, many nets need ripup and reroute. If a large k coefficient is applied, excessive routing wires would be rerouted with huge detour. While in the final stage of maze routing, the cost function curve is made steep to aggressively drive down the residual overflow. There are three other coefficients in the function: M is the cost when u_e is equal to c_e. S determines the slope when u_e is over c_e. H is the cost height which is increased each maze routing iteration. The parameters are experimentally determined.

$$cost_e = \begin{cases} 1 + H/(1 + exp(-k(u_e - c_e))) & \text{if} \quad 0 < u_e \leq c_e \\ 1 + M + S \times (u_e - c_e) & \text{if} \quad u_e > c_e \end{cases} \tag{5}$$

TABLE IV
EXPERIMENTAL BENCHMARKS STATISTICS

Name	Grids	#Nets	#Routed Nets	Max Deg	Avg Deg
ibm01	64×64	11.5k	9.1k	37	3.8
ibm02	80×64	18.4k	14.3k	126	4.4
ibm03	80×64	21.6k	15.3k	49	3.6
ibm04	96×64	26.2k	19.7k	41	3.4
ibm06	128×64	33.4k	25.8k	34	3.8
ibm07	192×64	44.4k	34.4k	22	3.8
ibm08	192×64	47.9k	35.2k	65	4.3
ibm09	256×64	50.4k	39.6k	38	3.8
ibm10	256×64	64.2k	49.5k	32	4.2
adaptec1	324×324	219k	177k	340	4.2
adaptec2	424×424	260k	208k	153	3.9
adaptec3	774×779	466k	368k	82	4.0
adaptec4	774×779	515k	401k	171	3.7
adaptec5	465×468	867k	548k	121	4.1
newblue1	399×399	332k	271k	74	3.5
newblue2	557×463	463k	374k	116	3.6
newblue3	973×1256	552k	442k	141	3.2
bigblue1	227×227	283k	197k	74	4.1
bigblue2	468×471	577k	429k	260	3.5
bigblue3	555×557	1.12M	666k	91	3.4
bigblue4	403×405	2.23M	1.13M	129	3.7
newblue4	455×458	636k	531k	152	3.6
newblue5	637×640	1.26M	892k	258	4.1
newblue6	463×464	1.29M	835k	123	3.8
newblue7	488×490	2.64M	1.65M	113	3.6

V. EXPERIMENTAL RESULTS AND COMPARISON

FastRoute3.0 is implemented in C, and all the experiments are performed on one 2.4Ghz Intel processor with 4GB of RAM. FLUTE [4] is utilized to generate RSMT. We demonstrate FastRoute3.0's performance by running three benchmark suites: ISPD98 benchmarks [13], 3D version of ISPD07 global routing contest benchmarks [14] and 3D version of ISPD08 global routing contest benchmarks [15]. The benchmark statistics are shown in table 4.

A. ISPD98 benchmarks

Table 5 shows the FastRoute3.0's performance for ISPD98 benchmarks. We make comparison with recently published academic global routers: NTHU-R, BoxRouter 2.0, Archer, FGR, FastRoute2.0 and BoxRouter. The results are quoted from [12], [8], [9], [10], [7] and [5] respectively. First, the result shows that FastRoute3.0 is able to route through all the benchmarks without any overflow. Second, FastRoute3.0 achieves good runtime. It can finish routing all 10 benchmarks within 15 seconds on our platform. Among all quoted global routers, FastRoute2.0 achieves fastest runtime. But it fails to generate congestion free solutions for ibm01, ibm04 and ibm09. Third, the total wirelength is comparable with Archer, NTHU-R, FGR and BoxRouter 2.0, which is 2.26% and 1.96% better than FastRoute2.0 and BoxRouter.

B. 3D version of ISPD07 benchmarks

Table 6 shows routing results on 3D version of ISPD07 global routing contest benchmarks. As shown in table 4, they are much harder in routing complexity and larger in grid size. We compare FastRoute3.0 with recently published global routers: FGR [10], MaizeRouter [11], BoxRouter 2.0 [8], FastRoute2.0 [7], Archer [9] and NTHU-R [12]. The first three routers are the winners of the 2007 ISPD global routing contest [14]. Speaking of final overflow, FastRoute3.0 is able to complete 7 benchmarks without any congestion. Noticeably, FastRoute3.0 successfully route through newblue1, which is not routable by any of the referred routers. For the unroutable one, newblue3, FastRoute3.0 produces the solution with lowest overflow. Secondly, in order to show FastRoute3.0's strong point on runtime, we run FGR 1.1 on our platform. The total runtime added together is around one hour,

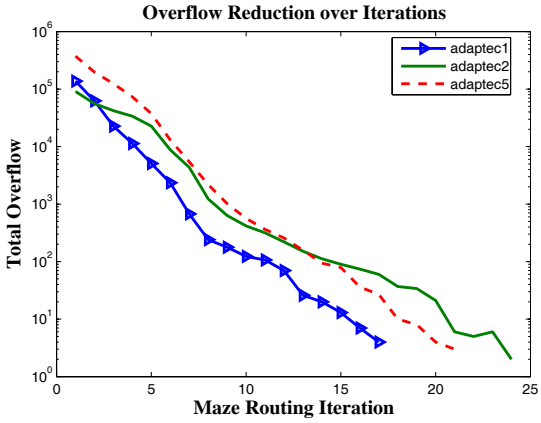

Fig. 7. FastRoute3.0 overflow reduction during maze routing iteration.

which is 64× faster than FGR 1.1. The runtime of Archer is also reported. From the literally quoted results [9], FastRoute3.0 is 9× faster. Note that Archer is performed on an Intel Xeon 3.60Ghz processor. Considering the wirelength, the reported results are calculated by the ISPD07 cost(segment wirelength plus three times of via number). FastRoute3.0 is comparable with all the winners of the contest, but it is much better than FastRoute2.0 with over 50% improvement. We also investigate the convergence of our router. Figure 8 shows that the total overflow of adaptec1, adaptec2, and adaptec5 goes down in logarithmical order and the required maze iteration is only around 20.

C. 3D version of ISPD08 benchmarks

In the ISPD 2008 global routing contest, 8 benchmarks are newly released. We evaluate FastRoute3.0's performance on the new benchmarks in table 7. The comparison is made with top 5 contest winners: NTHU-R, NTUgr, FastRoute3.0c [1], BoxRouter 2.0 and FGR. The contest results are obtained by running each submitted binary on a machine with up to four 2.8Ghz AMD processors[16]. First of all, in terms of overflow, FastRoute3.0 finishes routing 12 out of 16 benchmarks, which is the same as the best known results. Second, the total runtime added together is the second smallest. Although FastRoute3.0c achieves fastest runtime, it fails to route through newblue1 and bigblue2. Besides, NTUgr and FastRoute3.0c utilize multi-core programming, hence their single thread runtime would be slower. The runtime comparison may not be very accurate as the contest binaries are still not publicly available. Here we only want to demonstrate the runtime advantage of our router, which is at least comparable with the contest winners. Third, the ISPD08 wirelength cost(segment wirelength plus number of via) is on the same level with the others. It indicates that FastRoute3.0 doesn't sacrifice wirelength for achieving good runtime.

VI. CONCLUSIONS

In this paper, we have proposed FastRoute3.0, a fast and high quality global router with special emphasis on overflow reduction. The newly introduced technique is virtual capacity, which is used to guide the global router in maze routing stage out of local optimal solutions. FastRoute3.0 generates high quality solutions for ISPD98, ,ISPD07 and ISPD08 benchmark suites. But due to the fast growing problem size and degree of routing complexity, our future work will focus on two aspects. First, we will continue to improve FastRoute 3.0's routability and runtime. Second, we

[1]FastRoute3.0c is our contest version

TABLE V

COMPARISON OF FASTROUTE3.0, AND PUBLISHED GLOBAL ROUTERS ON ISPD98 BENCHMARKS.

name	FastRoute3.0			NTHU-R [12]			BoxRouer 2.0 [8]			FGR [10]			Archer [9]			FastRoute2.0 [7]			BoxRouter [5]		
	OF	wlen	cpu (sec)	OF	wlen	cpu (sec)	OF	wlen	cpu (sec)	OF	wlen	cpu (sec)	OF	wlen	cpu (sec)	OF	wlen	cpu (sec)	OF	wlen	cpu (sec)
ibm01	0	64221	0.64	0	63321	4.17	0	62659	33	0	63332	10	0	64389	11	31	68489	0.72	102	65588	8
ibm02	0	172223	0.85	0	170531	7.44	0	171110	36	0	168918	13	0	171805	25	0	178868	0.93	33	178759	34
ibm03	0	146753	0.49	0	146551	5.86	0	146634	18	0	146412	5	0	146770	10	0	150393	0.60	0	151299	17
ibm04	0	170146	2.70	0	168262	13.61	0	167275	116	0	167101	29	0	169977	24	64	175037	1.88	309	173289	24
ibm06	0	279471	1.15	0	278617	12.75	0	277913	47	0	277608	6	0	278841	23	0	284935	1.36	0	282325	33
ibm07	0	369023	1.68	0	366288	15.89	0	365790	86	0	366180	18	0	370143	25	0	375185	1.60	53	378876	51
ibm08	0	405935	1.82	0	405169	13.17	0	405634	90	0	404714	18	0	404530	42	0	411703	2.36	0	415025	93
ibm09	0	414913	1.67	0	415464	11.59	0	413862	273	0	413053	20	0	414223	37	3	424949	1.92	0	418615	64
ibm10	0	582838	3.61	0	580793	33.72	0	590141	352	0	578795	92	0	583805	45	0	595622	2.79	0	593186	95
Total	0	2606K	14.61	0	2595K	118.20	0	2601K	1051	0	2585K	211	0	2604K	242	98	2665K	14.16	497	2657K	419
Norm	/	1	1	/	0.996	8.09	/	0.998	72.94	/	0.992	14.44	/	1	16.56	/	1.023	0.97	/	1.020	28.68

TABLE VI

COMPARISON OF FASTROUTE3.0, AND PUBLISHED GLOBAL ROUTERS ON 3D VERSION OF ISPD07 GLOBAL ROUTING CONTEST BENCHMARKS

name	FastRoute3.0			FGR 1.1 [10]			Archer [9]			BoxRouter2.0 [8]		MaizeRouter [11]		FastRoute2.0 [7]	
	OF	wlen (e5)	cpu (min)	OF	wlen (e5)	cpu (min)	OF	wlen (e5)	cpu (min)	OF	wlen (e5)	OF	wlen (e5)	OF	wlen (e5)
adaptec1	0	92	5	0	88.5	345	0	114	87	0	92	0	100	122	249
adaptec2	0	93.4	1	0	90	32	0	113	23	0	94	0	98	500	244
adaptec3	0	205	4	0	200	200	0	244	51	0	207	0	214	0	523
adaptec4	0	188	1	0	179	29	0	222	12	0	186	0	194	0	469
adaptec5	0	271	11	0	260	748	0	334	248	0	270	0	305	9680	708
newblue1	0	94.5	5	314	94	1083	494	116	50	400	92.9	1348	102	1934	248
newblue2	0	136	1	0	129	9	0	167	7	0	135	0	140	0	380
newblue3	31634	182	36	45454	164	1513	31928	199	163	38958	172	32588	184	34236	443

TABLE VII

COMPARISON OF FASTROUTE3.0, AND ISPD08 GLOBAL ROUTING CONTEST RESULTS

name	FastRoute3.0			NTHU-R [16]			NTUgr [16]			FastRoute3.0c[16]			BoxRouter [16]			FGR [16]		
	OF	wlen (e5)	cpu (min)	OF	wlen (e5)	cpu (min)	OF	wlen (e5)	cpu (min)	OF	wlen (e5)	cpu (min)	OF	wlen (e5)	cpu (min)	OF	wlen (e5)	cpu (min)
adaptec1	0	55.2	5	0	53.5	8	0	56.1	5	0	55.5	2	0	53.8	20	0	54.1	35
adaptec2	0	53.7	1	0	52.3	2	0	53.4	1	0	53.1	1	0	52.7	3	0	52.6	14
adaptec3	0	133	4	0	131	8	0	134	5	0	133	2	0	132	27	0	132	55
adaptec4	0	123	1	0	122	2	0	123	2	0	122	1	0	122	7	0	122	17
adaptec5	0	161	11	0	156	17	0	159	17	0	161	5	0	157	31	0	157	110
newblue1	0	48.2	5	0	46.5	5	6	50.3	1150	76	49	13	44	47	1241	8	46.8	1412
newblue2	0	76.3	1	0	75.7	1	0	77.4	1	0	76.2	1	0	75.9	2	0	75.8	4
newblue3	31634	110	36	31454	106	129	31106	180	809	31650	109	31	32404	109	1377	34850	106	1427
bigblue1	0	59.5	7	0	56.3	10	0	57.8	14	0	58.3	4	0	57	19	0	57.3	70
bigblue2	0	98.8	16	0	90.6	10	0	97.2	264	142	98.2	11	0	90.4	39	0	91.4	238
bigblue3	0	132	2	0	131	4	0	136	5	0	132	3	0	131	6	0	132	88
bigblue4	156	244	35	182	231	126	188	243	413	206	243	41	472	232	877	414	232	1425
newblue4	154	137	17	152	130	67	142	144	1118	226	136	10	200	129	1304	262	130	1420
newblue5	0	240	11	0	232	14	0	246	28	0	241	5	0	233	28	0	233	166
newblue6	0	186	10	0	177	14	0	186	16	0	187	4	0	180	30	0	180	103
newblue7	108	361	160	68	354	141	310	372	1446	588	359	190	208	351	1412	1458	350	1434

will try to apply it in earlier physical design stage to produce routing friendly placement.

REFERENCES

[1] L. McMurchie and C. Ebeling, "Pathfinder: A negoriation-based performance-driven router for FPGAs", In *Proc. of Int'l Symp. on Field-Programmable Gate Arrays*, pp. 111-117, 1995.

[2] R. Kastner, E. Bozorgzadeh and M. Sarrafzadeh "Pattern Routing: Use and Theory for Increasing Predictability and Avoiding Coupling", In *Proc. of IEEE transactions on Computer-Aided Design of Integrated Circuits and Systems*, VOL. 21, NO. 7, July 2002.

[3] R.T. Hadsell and P.H. Madden, "Improved global routing through congestion estimation", In*Proc. of Design Automation Conference*, pp. 28-31, 2003.

[4] C. Chu, "FLUTE: Fast Lookup Table Based Wirelength Estimation Technique", In*Proc. of Intl. Conf. on Compuer-Aided Design*, pp. 696-701, Nov 2004.

[5] M. Cho and D. Z. Pan, "BoxRouter: A New Global Router Based on Box Expansion and Progressive ILP", In *Proc. of Design Automation Conference*, pp. 373-378, July 2006.

[6] M. Pan, and C. Chu, "FastRoute: A step to integrate global routing into placement", In *Proc. of Intl. Conf. on Computer-Aided Design*, pp. 464-471, Nov 2006.

[7] M. Pan, and C. Chu, "FastRoute2.0: A High-quality and Efficient Global Router", In *Proc. of Asia and South Pacific Design Automation Conf.*, Jan 2007.

[8] M. Cho, K. Lu, K. Yuan and D. Z. Pan, "BoxRouter 2.0: Architecture and Implementation of a Hybrid and Robust Global Router", In *Proc. of Intl. Conf. on Compuer-Aided Design*, pp. 503-508, Nov 2007.

[9] M. M. Ozdal and M. D.F. Wong "ARCHER: a history-driven global routing algorithm", In *Proc. of Intl. Conf. on Compuer-Aided Design*, pp. 481-487, Nov 2007.

[10] J. A.Roy and I. L. Markov, "High-Performance Routing at the Nanometer Scale", In *Proc. IEEE/ACM Intl. Conf. on Compuer-Aided Design*, pp. 496-502, Nov 2007.

[11] M. D. Moffitt, "MaizeRouter: Engineering an Effective Global Router", *Proc. Asia and South Pacific Design Automation Conf.*, Jan 2008.

[12] J.-R. Gao, P.-C. Wu, and T.-C. Wang "A New Global Router for Modern Designs", *Proc. Asia and South Pacific Design Automation Conf.*, Jan 2008.

[13] http://www.ece.ucsb.edu/∼kastner/labyrinth/.

[14] http://www.sigda.org/ispd2007/rcontest/.

[15] http://www.ispd.cc/contests/ispd08rc.html.

[16] http://www.ispd.cc/ispd08_technical_program.html.