A 3-Step Approach for Performance-Driven Whole-Chip Routing

Yih-Chih Chou

Department of Computer Sicence
National Tsing Hua University
Hsin-Chu, Taiwan 30043, R.O.C.
e-mail: dr834329@cs.nthu.edu.tw

Youn-Long Lin

Department of Computer Science
National Tsing Hua University
Hsin-Chu, Taiwan 30043, R.O.C.
e-mail: ylin@cs.nthu.edu.tw

Abstract

We propose a 3-step approach for whole-chip detail routing. In the first step, we construct a performance-driven Steiner tree for each net ignoring the existence of other nets. In the second step, we optimally assign significant wire segments of all trees to the tracks of a two-dimensional, two-layer grid under the design rule constraint. Finally, in the third step, we complete the remaining local short connection between net terminals and those assigned wire segments and resolve any violations or congestion. We have incorporated this approach into an industrial VDSM design flow. Experimental results on large benchmark circuits implemented in a TSMC 0.18 μm CMOS process have demonstrated the effectiveness of the proposed approach. We achieve more than 14% improvement over a state-of-art commercial performance-driven router in critical path delay. Our tool can be viewed as a preprocessor for a router. Users do not have to change their existing design flow. Only a small time-efficient step is needed to achieve the performance gain.

1 INTRODUCTION

In the early 1990s, the VLSI routing problem was considered to be solved. However, in large very deep submicron design, interconnect delay has dominated gate delay in total circuit delay. Therefore, we should re-examine the routing problem.

Many routing algorithms have been proposed for either performance-driven [2][3][5] or crosstalk reduction [1][4]. As six or more metal layers are available for routing, we should concentrate on how to use the routing resource efficiently for performance and reliability.

We propose a 3-step approach for whole-chip routing. In the first step, we use a performance-driven tree generator called TRIO [7] to construct an A-tree for each net ignoring the existence of other nets. In the second step, we assign all significant wire segments of all trees to the tracks of a two-dimensional, two-layer grid taking into account net criticality. Finally, in the third step, we complete the remaining local short connection between net terminals and those assigned wire segments.

The rest of this paper is organized as following. In section 2, we introduce some terminologies and formulate the problem. In section 3, we describe the proposed approach. Some experimental results are presented in section 4. Finally, in section 5 we draw some conclusions and point to possible directions for future research.

2 PROBLEM FORMULATION

Let's illustrate some terminologies with Fig. 1. N is a two-terminal net in the routing plane. Pin A and Pin B are two terminals of net N (Fig. 1(a)). A *wire segment* is an uninterrupted horizontal or vertical part of a routing tree of a net. For example, in Fig. 1(b) both segment AC and BC are wire segments of net N. The *netlength* of a net is the total length of all of its wire segments. A *track* is a pseudo straight line onto which wire segments can be assigned. In this example, the routing area consists of horizontal tracks h1, h2, ..., h5 and vertical tracks v1, v2, ..., v5. The separation between two adjacent tracks is called *pitch* and is defined by the process rule.

A wire segment can be assigned onto one of several possible tracks. An *assignment* is denoted as a pair (w, t) where t is the track onto which w is assigned. A *planning* of net n is defined by a set of assignments $\{(w_1, t_1), (w_2, t_2), \ldots, (w_m, t_m)\}$ where w_i is a wire segment of net n. For example, Fig. 1(c) and 1(d) show two different plannings of net N: P1 = {(AC, v3), (BC, h2)} and P2 = {(AC, v2), (BC, h1)}. Different plannings lead to different routing results. Fig. 1(e) and 1(f) shows the routing results of these two plannings, respectively.

We define the performance-driven net planning problem as: Given a placement result and the set of routing trees , find a planning for each net such that a performance-driven objective function is optimized under design rule constraints.

3 PROPOSED APPROACH

Our proposed approach consists of three steps: (1) performance-driven tree construction, (2) wire-segment-to-track assignment, and (3) local routing. Before describing the detail, let's illustrate them with an example depicted in Fig. 2. After placement, suppose we have two nets, $n1$(with terminals A and B) and $n2$(with terminals C, D, and E), as shown in Fig. 2(a). In the first step, we construct for each net independently a performance-driven tree (A-tree) and superimpose the routing region with a grid as depicted in Fig. 2(b). Note the design rule violations between wire segments $w1$ and $w2$. Fig. 2(c) shows the result after wire segments are assigned to the tracks of the grid. Note that the routing

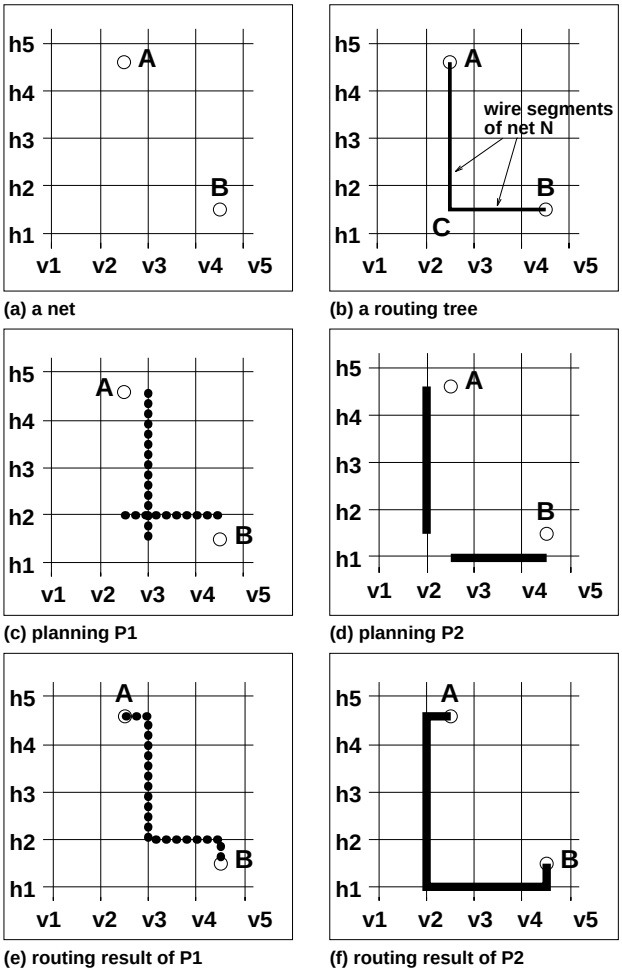

Fig. 1: Terminologies illustration

becomes incomplete because the wire segments have been shifted. There are 7 short local connections needed to be made. Fig. 2(d) depicts the result after the third step completes those short local connections.

A. *Tree Construction and Grid Superimposing*

Our first step constructs an A-tree for each net. Based on placement, we get the locations of all terminals of all nets. We construct an A-tree for each net using a public domain package called TRIO [7]. Nets are routed independent of one another. We will resolve any conflict later on.

After tree construction, a grid is superimposed on the layout area. It consists of two metal layers in different direction for signal routing (e.g., Metal5 for horizontal tracks and Metal6 for vertical tracks in a modern 0.18 μm CMOS process). The grid pitch is determined by the design rule.

B. *Wire-segment-to-track Assignment*

A pseudo code description of our algorithm is given in Fig. 3 and 4. As illustrated in Fig. 2(b), the wire segments of the routing trees are not aligned on the grid tracks and there may exist congestion and design rule violation among trees.

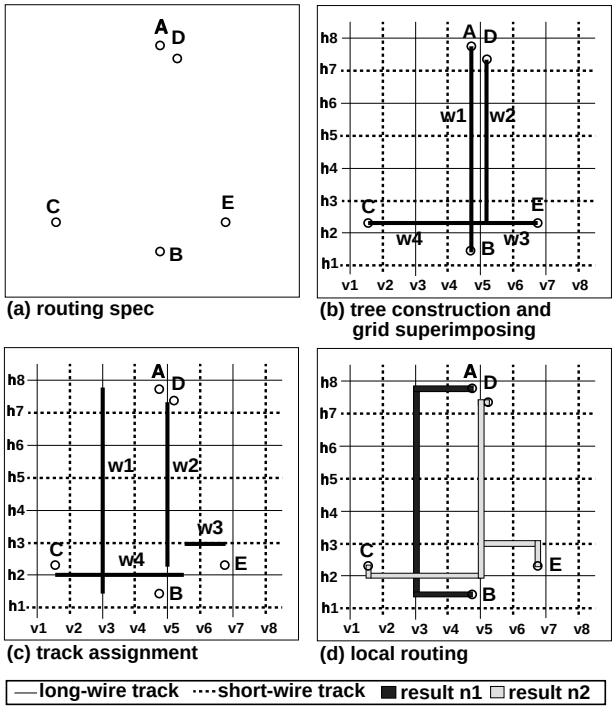

Fig. 2: An illustrative example

In this step, we assign *significant* wire segments to tracks. A segment is significant if it is longer than a user-given threshold value S_{th}. The remaining wire segments will be connected during the last step.

In order to reduce coupling capacitance, we do not allow two long wire segments to run next to each other. Significant wire segments are categorized into long and short categories. A segment is long if it is longer than another user-specified threshold parameter L_{th}; otherwise, it is a short segment. Tracks are alternately designated as *long-wire-tracks* (solid lines in Fig. 2(b)) and *short-wire-tracks* (dashed lines). A long segment can only be assigned to a long-wire-track while a short segment can only be assigned to a short-wire-track.

A wire segment w may be assigned to one of a set of target tracks. This set, $TT(w)$, is determined by a user-defined range R. For example, for R=3, a vertical long wire segment may be assigned to one of six vertical long-wire-tracks, three immediately to its left and three immediately to its right. For each track t in $TT(w)$, we have to compute the cost function $f(w,t)$ of assigning wire segment w of net n to track t as

$$f(w,t) = \frac{Distance(w,t)}{Length(n) \cdot Criticality(n)}$$

where $Distance(w,t)$ is the distance between wire segment w and track t, $Length(n)$ is the netlength of net n, and $Criticality(n)$ is the timing slack of net n. The smaller the f value is, the more preferable the assignment is.

Now we have to decide exactly which wire segment will be assigned to which track. First, for each wire segment w, we will sort tracks in $TT(w)$ into ascending order according to their cost function values. Then we put all wire segments into a queue ($Wire\_list$). We perform the wire-segment-to-track assignment iteratively. Once a wire segment w is

Algorithm PNP

```
begin
  for each net n_i do
    import a_i, the A-Tree of n_i;
  end for
  construct grid;
  for each a_i do
    for each wire segment w_j of a_i do
      if (length(w_j) > S_th) then
        for each track t_k ∈ TT(w_j) do
          compute f(w_j, t_k);
        end for
        sort TT(w_j) according to f values;
        w_j.current_track ← first track of TT(w_j);
        w_j.next_track ← second track of TT(w_j);
        enqueue(Wire_list, w_j);
      end if
    end for
  end for
  while Wire_list ≠ φ do
    w = dequeue(Wire_list);
    Insert(w, w.current_track);
  end while
end
```

Fig. 3: The performance-driven net planning(PNP) algorithm

removed from the queue, we check the assignment in its currently most preferable track ($w.current\_track$). If no violation occurs, w is assigned to $w.current\_track$. Otherwise, w should compete with every wire segment w_i which have been assigned to the track and overlapped with w. If w wins (i.e., w's f value is smaller than all of its competitors), we assign it to the track, de-assign all its competitive segments from the track and try inserting each of them to its next most preferable tracks. If w loses (i.e., w's f value is larger than any of its competitors), we should try its next most preferable track ($w.next\_track$). For example, in Fig. 2(c), because wire segment $w1$ fails in competing with $w2$ for track $v5$, it is assigned to track $v3$, its secondary choice. If there is no more track on $TT(w)$, we will give up w and let the third step to route it. After all wire segments in the queue $Wire\_list$ are processed, the assignment is very effective in terms of the total cost function value of all assignment. According to our algorithm, we have the following conclusion:

Theorem 3.1 *The final planning result is independent of the order of the planning sequence.*

<proof> Suppose i is the length of the planning sequence. For $i = 1$, there exists only one sequence $\{w\}$, it is true. Suppose the theorem holds for i = n - 1. For i = n, suppose the planning sequence is $\{w_1, w_2, \ldots, w_{n-1}, w_n\}$, we can divide this sequence into two subsequences $\{ w_1, w_2, \ldots, w_{n-1}\}$ and $\{w_n\}$. Choose one element w_k from the first subsequence. By the induction hypothesis, the planning sequence $\{w_1, w_2, \ldots, w_{k-1}, w_{k+1}, \ldots, w_{n-1}, w_k\}$ will generate the same planning result as the sequence $\{w_1, w_2, \ldots, w_{n-1}\}$ do. Thus the planning sequence $\{w_1, w_2, \ldots, w_{k-1}, w_{k+1}, \ldots, w_{n-1}, w_k, w_n\}$ will generate the same result as the original sequence do. Let's change this sequence by swapping w_k and w_n. The new sequence is $\{w_1, w_2, \ldots, w_{k-1}, w_{k+1}, \ldots, w_{n-1}, w_n, w_k\}$. Because w_n and w_k are the last two segments, it is trivial their order will not affect the planning result. Therefore the new sequence will also produce the same result as the original one do. Now we can divide the new sequence into two subsequences $\{w_1, w_2, \ldots, w_{k-1}, w_{k+1}, \ldots, w_{n-1}, w_n\}$ and $\{w_k\}$. By the induction hypothesis, no matter how w_n is placed in the first subsequence, it will produce the same result. Thus the extended sequence (with last element w_k) will produce the same result as the original sequence do. By the statements above, the position of w_n in the planning sequence with length n will not affect the planning result.
By M.I.D., the final planning result of each wire segment w_n is independent of the position of w_n in the planning sequence. □

Procedure Insert(w, t)

```
begin
  if t has no assigned wire segment overlap with w then
    PL(t) ← PL(t) + w;
    // PL(t) is the list of all wire segments assigned to t;
  else
    W ← {w' | w' overlaps with w on t};
    for each w'' ∈ W do
      if f(w,t) > f(w'',t) then
        if w.next_track ≠ φ then
          w.current_track ← w.next_track;
          w.next_track ← first untouched track in
            TT(w);
          Insert(w, w.current_track);
        end if
        exit;
      else if f(w,t) = f(w'',t) then
        if w.index > w''.index then
        // w.index is the order of w in the planning
          sequence;
          if w.next_track ≠ φ then
            w.current_track ← w.next_track;
            w.next_track ← first untouched track in
              TT(w);
            Insert(w, w.current_track);
          end if
          exit;
        end if
      end for
      PL(t) ← PL(t) − W;
      PL(t) ← PL(t) + w;
      for each w'' ∈ W do
        if w''.next_track ≠ φ then
          w''.current_track ← w''.next_track;
          w''.next_track ← first untouched track in
            TT(w'')
          Insert(w'', w''.current_track);
        end if
      end for
    end if
end
```

Fig. 4: The wire-segment-to-track assignment procedure

Table 1: Benchmark Specifications

Benchmark	# cells	# nets	# I/O	area(μm^2)
addrgen	1031	1161	96	102713
rgb_intr	2619	2778	71	188191
matrix	3375	3603	119	227405
sdram_rdr	4125	4559	95	365698
32bMAC	8655	8941	213	562695
VP2	10063	10542	323	657251
a259k	95765	104683	153	4392336
a518k	191592	209354	153	8230874

Table 2: Experimental Results

Benchmark	Critical Path Delay			CPU
	TRA(ns)	PNP(ns)	imprv%	Time(s)
addrgen	2.12	1.99	5.76	0.41
rgb_intr	6.86	6.23	9.16	1.22
matrix	8.42	8.17	2.97	1.39
sdram_rdr	2.81	2.67	5.10	3.64
32bMAC	4.85	4.23	12.83	12.86
VP2	13.66	12.61	7.66	11.02
a259k	12.35	10.68	13.52	891.03
a518k	14.26	12.25	14.13	3402.61

C. *Local Routing*

The last step completes the remaining connections. For each net, this step will complete the connection between net terminals and assigned wire segments. We rely on the router in the existing design flow to accomplish this step. Fig. 2(d) gives an example of this process. All vacant space in all available layers are utilized.

4 EXPERIMENTS

We have implemented the proposed algorithm in a C++ program running on a SUN UltraSparc 80 workstation. The experimental setup is illustrated in Fig. 5. First, we read the RTL-level benchmark and do technology mapping using Synopsys's Design Analyzer. Second, we import the mapping result to Cadence's Silicon Ensemble Ultra for floorplanning and timing-driven placement. Then we construct a performance-optimized A-tree for each net using TRIO [7], perform our wire-segment-to-track assignment(PNP), and export the partially routed result in the DEF [12] format for Silicon Ensemble Ultra to complete the remaining routing. We achieve the local routing step by the *incremental* wrap route process. Finally, we extract the parasitic RC values using Cadence's HyperExtract and back-annotate the cell/net delay information in the SDF [10] format to Design Analyzer for critical path timing analysis.

We also run the traditional flow(TRA) for comparison. The **timing-driven wrap route** process of Silicon Ensemble Ultra is employed in this flow.

We use seven benchmark designs to evaluate our approach. Their specifications are given in Table 1. The cell counts range from 1K to 191K gates in a TSMC 0.18 μm CMOS cell library from Artisan [11]. The threshold values S_{th} and L_{th} are chosen according to the chip area. We give

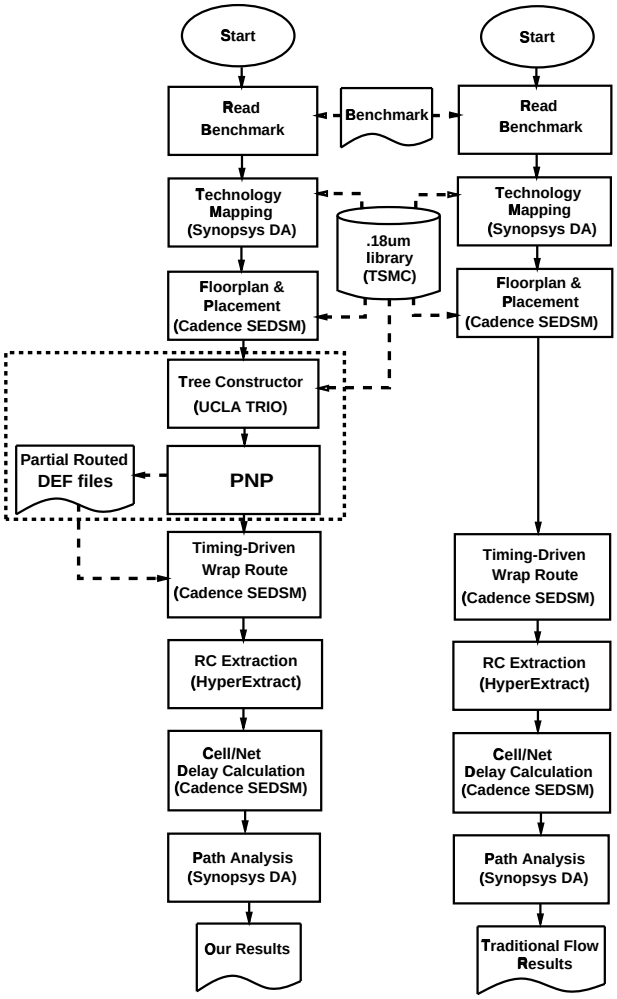

Fig. 5: Experimental flow

the critical path delay values in Table 2. The critical path delay is calculated using Synopsys's Static Timing Analyzer (DesignTime). The critical path delay improves by up to 14% in the largest design. The CPU time consumption is very small compared with other tasks of the flow.

5 CONCLUSIONS AND FUTURE WORK

We have proposed a 3-step approach for whole-chip detail routing. By globally, optimally assigning significant wire segments to routing tracks, we are able to preserve as much the performance-optimal tree topology constructed by an A-tree algorithm as possible. Experimental results have demonstrated that our approach is indeed very effective in resolving the conflicts among nets (wire segments) competing for resources (tracks). Moreover, our tool can be used as a preprocessor to any traditional router. Thus, it is very easy to be incorporated into an existing design (APR) flow. The performance gain is significant while the investment is small.

In the future, we would like to extend this work in two directions: crosstalk reduction and obstacle avoidance. By alternatively designated tracks for long and short wire seg-

ments, we have intuitively prevented pairs of long wires from running in parallel. However, we have to study in more detail the effectiveness of this approach in crosstalk reduction. For obstacle avoidance, the wire-to-track assignment procedure can be modified by limiting the set of possible target tracks. The biggest issue lies in constructing the A-tree under the obstacle constraint. Currently, TRIO does not have this capability. We have to either enhance it or find another package with the capability.

References

[1] H. H. Chen and C. K. Wong, "Wiring and crosstalk avoidance in multi-chip module design," in *Proc. Custom Integrated Circuits Conf.*, pp. 28.6.1-28.6.4, 1992.

[2] J. Cong, K. S. Leung, and D. Zhou, "Performance-driven interconnect design based on distributed RC delay model," *Proc. IEEE Design Automation Conf.* , pp. 606-611, 1993.

[3] W. M. Dai, R. Kong, J. Jue, and M. Sato, "Rubber band routing and dynamic data representation," *Proc. Int. Conf. Computer-Aided Design*, , pp. 52-55, 1990.

[4] T. Gao and C. L. Liu, "Minimum crosstalk channel routing," *Proc. Int. Conf. Computer-Aided Design*, , pp. 692-696, 1993.

[5] Jens Lienig, "A Parallel Genetic Algorithm for Performance-Driven VLSI Routing," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 29-39, April, 1997.

[6] Joon-Seo Yim and Chong-Min Kyung, "Reducing cross-coupling among interconnect wires in deep-submicron datapath design," *Proc. IEEE Design Automation Conf.* , pp. 485-490, 1999.

[7] Lei He, Cheng-Kok Koh, David Z. Pan, Xin Yuan and Jason Cong, "TRIO: A Tree, Repeater and Interconnect Optimization Package," http://cadlab.cs.ucla.edu/~trio.

[8] H. B. Bakoglu, "Circuits, interconnections and packaging for VLSI," Addison-Welesy, 1990.

[9] Sabih H. Gerez, "Algorithms for VLSI Design Automation," John Wiley & Sons, 1999.

[10] Pran Kurup and Taher Abbasi, "Logic Synthesis Using Synopsys," Second Edition, Kluwer Academic Publishers, 1998.

[11] Artisan Components Inc., http://artisan.com.

[12] Cadence Design Systems, Inc., "LEF/DEF Language Reference," Product Version 5.0, February 1997.