TILA-S: Timing-Driven Incremental Layer Assignment Avoiding Slew Violations

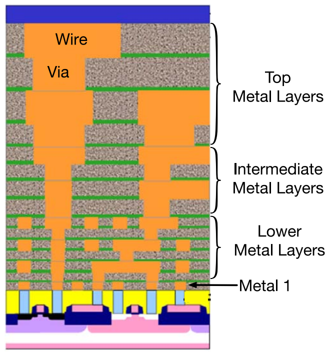
Derong Liu, Bei Yu, *Member, IEEE*, Salim Chowdhury, and David Z. Pan, *Fellow, IEEE*

***Abstract*—As very large scale integration technology scales to deep submicrometer and beyond, interconnect delay greatly limits the circuit performance. The traditional 2-D global routing and subsequent net by net assignment of available empty tracks on various layers lacks a global view for timing optimization. To overcome the limitation, this paper presents a timing driven incremental layer assignment tool, to reassign layers among routing segments of critical nets and noncritical nets. Lagrangian relaxation techniques are proposed to iteratively provide consistent layer/via assignments. Modeling via min-cost flow for layer shuffling avoids using integer programming and yet guarantees integer solutions via uni-modular property of the inherent model. In addition, multiprocessing of $K \times K$ partitions of the whole chip provides runtime speed up. Furthermore, a slew targeted optimization is presented to reduce the number of violations incrementally through iteration-based Lagrangian relaxation, followed by a post greedy algorithm to fix local violations. Certain parameters introduced in the models provide tradeoff between timing optimization and via count. Experimental results in both ISPD 2008 and industry benchmark suites demonstrate the effectiveness of the proposed incremental algorithms.**

***Index Terms*—Global routing, layer assignment, min-cost network flow, timing.**

I. Introduction

As very large scale integration technology scales to deep submicrometer and beyond, interconnect delay plays a determining role in timing [1]. Therefore, interconnect synthesis, including buffer insertion/sizing and timing-driven routing, becomes a critical problem for achieving timing closure [2]. Global routing is an integral part of a timing convergence flow to determine the topologies and layers of nets, which greatly affect the circuit performance [3]–[9]. In emerging technology nodes, back-end-of-line metal stack offers heterogeneous routing resources, i.e., dense metal at the lower layers and wider pitches at the upper layers. Fig. 1 gives one example of cross section of IC interconnection stack in advanced technology nodes [10], where wires and vias on top metal layers are much wider and much less resistive than those on lower metals. Besides, the normalized pitches of different metal layers from [11] are also listed. Advanced routing algorithms should not only be able to achieve routability, but also intelligently assign layers to overcome interconnect timing issues.



Layer	Pitch
M10	1.00
M9	1.00
M8	0.63
M7	0.63
M6	0.63
M5	0.63
M4	0.63
M3	0.33
M2	0.35
M1	0.33

Fig. 1. Cross section of IC interconnection stack in advanced technology nodes [10], where wires and vias on top metal layers are much wider and much less resistive than those on lower metals. The normalized pitch lengths of different metal layers are listed in the table (source: [11]).

Layer assignment is an important step in global routing to assign each net segment to a metal layer. It is commonly generated during or after the wire synthesis to meet tight frequency targets, and to reduce interconnect delay on timing critical paths [12]. In layer assignment, wires on thick metals are much wider and thus, less resistive than those on thin metals. If timing critical nets are assigned to lower layers, it will make timing worse due to narrower wire width/spacing. Although top metal layers are less resistive than those in lower (thin) metals, it is impossible to assign all wires to top layers. That is, layer assignment should satisfy the capacity constraints on metal layers. If an excessive number of wires are assigned to a particular layer, it will aggravate congestion and crosstalk. Meanwhile, the delay due to vias cannot be ignored in emerging technology nodes [1]. In addition, during timing closure slew violations could affect the utilization of buffering resources [13]. Thus, to guarantee signal integrity and reduce buffering resources, slew violations need to be avoided during layer assignment.

Recently, layer assignment has been considered in two design stages, i.e., buffered tree planning and 3-D global routing. Some studies consider layer assignment during buffer routing trees design [12], [14], [15]. Li *et al.* [12] proposed a set of heuristics for simultaneous buffer insertion and layer assignment. Hu *et al.* [14], [15] proved that, even if buffer positions are determined, the layer assignment

Manuscript received December 31, 2015; revised May 22, 2016 and October 26, 2016; accepted December 15, 2016. Date of publication January 11, 2017; date of current version December 20, 2017. This work was supported in part by the Oracle Laboratory, in part by the National Science Foundation, in part by SRC, and in part by the CUHK Direct Grant for Research. A preliminary version has been presented at the International Conference on Computer-Aided Design (ICCAD) in 2015. This paper was recommended by Associate Editor Z. Li. *(Corresponding author: Derong Liu.)*

D. Liu and D. Z. Pan are with the Department of Electrical and Computer Engineering, University of Texas at Austin, Austin, TX 78712 USA (e-mail: derongliuliu@gmail.co).

B. Yu is with the Department of Computer Science and Engineering, Chinese University of Hong Kong, Hong Kong.

S. Chowdhury is with Oracle Corporation, Austin, TX 78727 USA.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2017.2652221

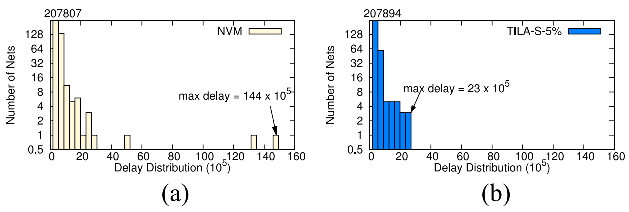


Fig. 2. Net delay distribution for benchmark `adaptec2`. (a) Result by layer assignment solver NVM [18]. (b) Result by our TILA-S, where 5% most critical nets are reassigned layers.

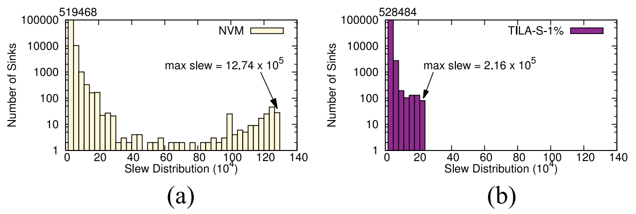


Fig. 3. Sink slew distribution for benchmark `adaptec2`. (a) Result by layer assignment solver NVM [18]. (b) Result by our TILA-S, where 1% most critical nets are reassigned layers.

with timing constraints is $\mathcal{NP}$-complete. During 3-D global routing, layer assignment is a popular technique for via minimization. Cho and Pan [3] proposed an integer linear programming (ILP)-based method to solve the layer assignment problem. Since via minimization is the major objective, all wires tend to be assigned onto the lower layers. Lee and Wang [16] and Dai *et al.* [17] applied dynamic programming to solve optimal layer assignment for a single net. To overcome the impact of net ordering, different heuristics or negotiation techniques were proposed in [18] and [19]. Ao *et al.* [19] considered the delay in layer assignment, but since via capacity was not considered, more segments can be illegally pushed onto higher routing layers. A min-cost flow-based refinement was developed in [20] to further reduce the number of vias. Furthermore, Lee *et al.* [21] proposed an enhanced global router with layer assignment refinement to reduce possible violations. Recently, Liu *et al.* [22], [23] solved the layer assignment problem through semidefinite model with a more accurate calculation of via costs; and Livramento *et al.* [24] targeted at optimizing timing paths through a network model. For slew optimization, repeaters/buffers insertions are widely adopted to fix the potential slew violations [12], [13], [25]. Zhang *et al.* [26] utilized an ILP approach to reconstruct the over-the-block steiner tree structure to improve slew.

Existing layer assignment studies suffer from one or more of the following limitations.

1) Most works only target at via number minimization, but no timing issues are considered. Since timing requirements within a single net are usually different for different sinks, assigning all segments of a set of nets on higher metal layers is not the best use of critical metal layer resources. That is, intelligent layer assignment should not blindly assign all segments of a net to a set (a pair, for example) of higher metal layers. It should be aware of capacitive loading of individual segments within a net to achieve better timing with the limited available higher metal layer resources.
2) In emerging technology nodes, the via delays contribute a non-negligible part of total interconnect delay. But the delay impact derived from vias is usually ignored in previous layer assignment works.
3) During post routing stage, slew violations may cause buffering resources. There are limited works to avoid slew violations globally during layer assignment.
4) The net-by-net strategy may lead to local optimality, i.e., for some nets the timings are over-optimized, while some other nets may have no enough resources in high layers. Meanwhile, considering one edge at each time may lose potential optimality because the edge ordering could also affect the subsequent solutions.

To close on timing for critical nets that need to go long distances, layer assignment needs to be controlled by multinet global optimization. For example, Fig. 2 compares the delay distributions of benchmark "`adaptec2`" by conventional layer assignment solver [18] and our incremental timing-driven solution, while Fig. 3 compares the slew distribution results. We can see that, since conventional layer assignment only targets at via minimization, the maximum delay and the maximum slew can be very large. Since our timing-driven planner is with global view, the maximum delay can be much better, i.e., the normalized maximum delay can be reduced from 144×10^5 to 23×10^5. Meanwhile, the slew violations can also be reduced significantly. The maximum slew decreases from 12.74×10^5 to 2.16×10^5.

For very large high-performance circuits, either long computation times have to be accepted or routing quality must be compromised. Therefore, an incremental layer assignment to iteratively improve routing quality is a must. In this paper, we propose an incremental layer assignment framework targeting at timing optimization. Incremental optimizations or designs are very important in physical design and CAD field to achieve good timing closure [27]. Fast incremental improvements are developed in different timing optimization stages, such as incremental clock scheduling [28], [29], incremental buffer insertion [30], and incremental clock tree synthesis [31]. To further improve timing, incremental placement is also a very typical solution [32], [33]. Besides, there are several incremental routing studies (e.g., [34]) to introduce cheap and incremental topological reconstruction.

To the best of our knowledge, this paper is the first incremental layer assignment work integrating via delay and solving all the nets simultaneously. A multilayer global router can either route all nets directly on multilayer solution space [4], [5] or 2-D routing followed by post-stage layer assignment [6]–[9]. Note that as an incremental layer assignment solution, our tool can smoothly work with either type of global router. Our contributions are highlighted as follows.

1) A mathematical formulation gives the layer assignment solutions with optimal total wire delays and via delays.
2) A Lagrangian relaxation-based optimization iteratively improves the layer assignment solution.
3) Lagrangian relaxation subproblem (LRS) is solved via min-cost flow model that guarantees integer solutions due to inherent uni-modular property, thus, avoiding runtime extensive methods, such as ILP.
4) An iterative Lagrangian relaxation-based slew optimization strategy is proposed to reduce the violations globally.
5) A post slew optimization algorithm searches potential usable layers for fixing local violations.
6) Multiprocessing of $K \times K$ partitions of the whole chip provides runtime speed up.

The remainder of this paper is organized as follows. Section II provides some preliminaries and the problem formulation. Section III gives mathematical formulation, and

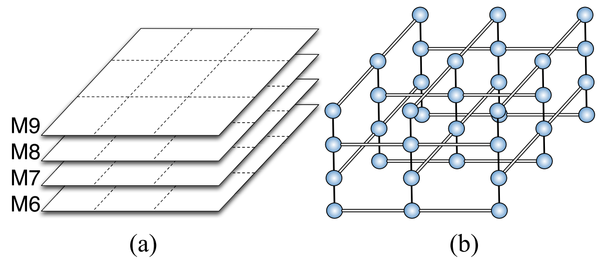


Fig. 4. Layer design and grid models. (a) Design with four routing layers $\{M6, M7, M8, M9\}$. (b) Grid model with preferred routing directions.

also proposes sequence of multithreaded min-cost flow algorithm to achieve further speed-up. In addition, mitigating slew violations is discussed in this section. Section IV reports experimental results, followed by the conclusion in Section V.

II. Preliminaries and Problem Formulation

In this section we introduce the graph model and the timing model applied in this paper. Then the problem formulation of timing-driven incremental layer assignment (TILA) is provided.

A. Graph Model

Similar to the 3-D global routing problem, layer assignment problem can be modeled on a 3-D grid graph, where each vertex represents a rectangular region of the chip, so called a global routing cell (G-Cell), while each edge represents the boundary between two vertices. In the presence of multiple layers, the edges in the z-direction represent vias connecting different layers. Fig. 4(a) shows a grid graph for routing a circuit in multimetal layer manufacturing process. Each metal layer is dedicated to either horizontal or vertical wires. The corresponding 3-D grid graph is shown in Fig. 4(b).

To model the capacity constraint, for each x/y-direction edge, we denote its maximum routing capacity as c_e. Besides, the via capacity of each vertex, denoted by c_v, is computed as in [35]. In brief, via capacity refers to the available space for vias passing through the cell, and is determined by the available routing capacity of those two x/y-direction edges connected with the vertex. Thus, this via capacity model helps to keep adequate routing space for vias through layers, and places the limits of wires on higher metal layers, which may result in wire delay degradation.

B. Delay Model

We are given a global routing of nets, where each net is a tree topology with one source and multiple sinks. Based on the topology, for each net we have a set of segments S. Here, we give an example of net model in Fig. 5, where each net contains two segments. To evaluate the timing of each net, we adopt *Elmore* delay model, which is widely used during interconnect synthesis in physical design. The delay of a segment s_i on a layer l, denoted by $d_e(i, l)$, is computed as follows:

$$d_e(i, l) = R_e(l) \cdot (C(l)/2 + C_{\text{down}}(s_i)) \tag{1}$$

where $R_e(l)$ and $C(l)$ refer to the edge resistance on layer l, and edge capacitance on layer l, respectively. $C_{\text{down}}(s_i)$ refers to the downstream capacitance of s_i. Note that the downstream capacitance of s_i is determined by the assigned layers of its all downstream segments. To calculate the downstream capacitance for each s_i, we should traverse the net tree from sinks

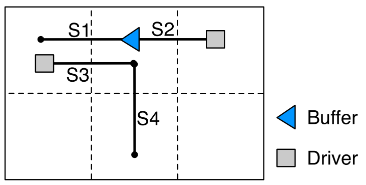


Fig. 5. Example of net model.

to source in a bottom-up manner. Therefore, the downstream capacitance of the source segment, i.e., the segment connected with the driver pin, should be calculated after all the other segments have obtained their downstream capacitances.

For a via v_m connecting segments between layers l and $l+1$, its delay can be calculated as follows:

$$d_v(v_m, l) = R_v(l) \cdot C_{\text{down}}(v_m). \tag{2}$$

Here, $R_v(l)$ is the resistance of via between layers l and $l+1$, and $C_{\text{down}}(v_m)$ is the downstream capacitance of the upstream segment connected to via v_m. If the downstream capacitance of a via is equal to zero, then we assume the via delay is negligible.

In addition, buffering can be considered in our delay model. As shown in Fig. 5, $C_{\text{down}}(s_2)$ is equal to the input capacitance of the buffer. Because buffers are fabricated in silicon and have pins connected with a specified metal layer, integration with buffers would affect the downstream capacitance for the corresponding pin. Meanwhile, buffering would also introduce buffer intrinsic delay and driving delay for each driving net. The intrinsic delay is dependent on the driving buffer, while the driving delay is in proportion to the downstream capacitance. Because capacitances of different layers vary less than resistances, we do not include the buffer driving delay in this paper. Therefore, through updating the downstream capacitances and including buffer intrinsic delay, our framework can handle timing optimization for both prebuffered and post-buffered designs.

C. Slew Model

Besides delay, our framework also considers slew computation to reduce the potential slew violations. Since each routing net is a tree topology in essence, we traverse the tree in a breadth-first manner from the driver to each sink and calculate the slew for each pin. For each segment, the input slew is represented by its upstream pin slew, and the output slew by its downstream pin slew. To calculate the output slew, we adopt *PERI* model, which has been shown to provide less than 1% error [36]. The calculation is given in (3), where $\text{Slw}(p_u(s_i))$, $\text{Slw}(p_d(s_i))$ are the input and output slew of s_i, respectively, while $\text{Slw}_{\text{step}}(s_i)$ is the step slew

$$\text{Slw}(p_d(s_i)) = \sqrt{\text{Slw}(p_u(s_i))^2 + \text{Slw}_{\text{step}}(s_i)^2}. \tag{3}$$

Based on *PERI* model, the segment output slew depends on both its input slew and step slew. The input slew is also the output slew of the upstream segment, so it can be obtained iteratively through (3). Regarding the step slew, we calculate it through the combination of *PERI* model and *Bakoglu's metric*. It is proved to have error within 4% [36]. The calculation is shown in (4), where $l(s_i)$ is the layer on which s_i is assigned, and $d_e(i, l)$ is *Elmore* delay of segment s_i on layer l

$$\text{Slw}_{\text{step}}(s_i) = \text{Slw}_{\text{step}}(i, l(s_i)) = \ln 9 \cdot d_e(i, l(s_i)). \tag{4}$$

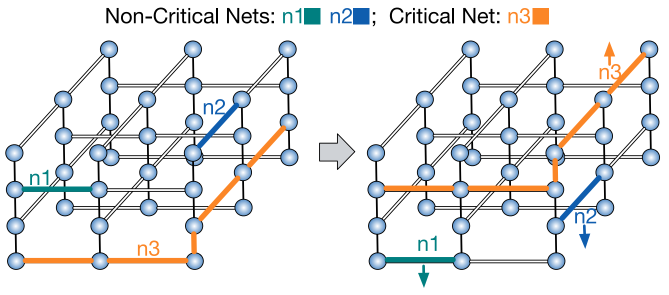


Fig. 6. Example of timing driven layer assignment. In initial layer assignment net n_3 is timing critical net. Through resource releasing from nets n_1 and n_2, the total timing gets improvement.

With the calculated step slew, we can obtain the output slew for each segment. To see the impact of layer assignment, the output slew can be represented as a function of its input slew and the layer to be assigned

$$\mathrm{Slw}_e(i, l(s_i)) = \sqrt{\mathrm{Slw}(p_u(s_i))^2 + (\ln 9 \cdot d_e(i, l(s_i)))^2}. \quad (5)$$

Besides, via slew should also be considered during slew calculation and computed in a similar way as segment slew. Equation (6) gives the slew for via v_m from layer l to layer $l+1$

$$\mathrm{Slw}_v(v_m, l+1) = \sqrt{\mathrm{Slw}(p_{v_m})^2 + (\ln 9 \cdot d_v(v_m, l))^2}. \quad (6)$$

In contrary to downstream capacitance calculation in a bottom-up manner, here we start from the segment connected with the net driver. Then each segment and its connected via are traversed in a breadth-first manner until every sink is reached. With this approach, we obtain the output slew for each net sink sequentially. If the sink slew exceeds a specified slew constraint, we assume there is a slew violation.

D. Problem Formulation

Based on the grid model and timing model discussed in the preceding section, we define the TILA problem as follows.

*Problem 1 (*TILA*):* Given a global routing grid, a set of critical net segments and layer/via capacity information, TILA assigns each segment passing through an edge to a layer, so that layer assignment costs (weighted sum of segment delays, via delays, and slew violations) can be minimized, while the capacity constraints of each edge on each layer are satisfied.

One instance of TILA problem with three nets is demonstrated in Fig. 6, where nets n_1 and n_2 are noncritical nets, while net n_3 is timing critical. Initially, net n_3 is assigned to lower layers. Since the routing resources are utilized by nets n_1 and n_2, n_3 cannot be shuffled into higher layers to improve timing. Through a global layer reassignment, we are able to achieve a better timing assignment solution, where both n_1 and n_2 release high layer resources to n_3.

Naclerio *et al.* [37] proved that even if no timing is considered, the decision version of layer assignment for via minimization is $\mathcal{NP}$-complete. Thus, the decision version of TILA problem is $\mathcal{NP}$-complete as well.

TABLE I
NOTATIONS USED IN THIS PAPER

L	number of layers
S	set of all segments considered
E	set of all edges
G	set of all g-cells on 2-D plane
E_x	set of all pairs of crossing segments
$P(s_i)$	nodes of segment s_i, i.e. upstream pin and downstream pin
$N(v_m)$	set of neighboring segments of via v_m
$S_e(i)$	set of segments assigned to the same edge as s_i
$E_x(g)$	set of crossing segment pairs passing through g-cell g
a_{ij}	binary variable; if i-th segment is assigned to layer j then $a_{ij} = 1$, otherwise $a_{ij} = 0$
$d_e(i,j)$	timing cost if s_i is assigned to layer j
$d_v(i,p,k)$	timing cost of via v from layer k to $k+1$, where $v \in P(s_i) \cap P(s_p)$
$l(s_i)$	layer where segment s_i is assigned
$c_e(i,j)$	routing capacity of edge e through which s_i passes on layer j
$c_g(k)$	available via capacity of g-cell g on layer k

III. TILA ALGORITHMS

In this section, we introduce our framework to solve the TILA problem. First a mathematical formulation targeting delay optimization will be given. Then a Lagrangian relaxation-based optimization methodology is proposed to solve this problem. After the delay optimization, a Lagrangian relaxation-based slew optimization is presented, followed by a post optimization stage. For convenience, some notations used in this section are listed in Table I.

A. Mathematical Formulation

The starting mathematical formulation of TILA problem is shown in (7). In the objective function, the first term is to calculate the cost from segments, while the second term is to calculate the cost from vias. Here, $d_e(i,j)$ is calculated through (1), and $d_v(i,p,k)$ is derived from (2)

$$\begin{aligned}
\min\ & \sum_{i\in S}\sum_{j=1}^{L} d_e(i,j)\cdot a_{ij} \\
& + \sum_{(i,p)\in E_x}\sum_{j=1}^{L}\sum_{q=1}^{L}\sum_{k=\min(j,q)}^{\max(j,q)-1} d_v(i,p,k)\cdot a_{ij}\cdot a_{pq} && \text{(7a)}\\
\text{s.t. } & \sum_{j} a_{ij} = 1, \forall i \in [1,S] && \text{(7b)}\\
& \sum_{s_i\in S_e(i)} a_{ij} \le c_e(i,j), \forall e\in E, \forall j\in[1,L] && \text{(7c)}\\
& \sum_{(i,p)\in E_x(g)}\ \sum_{\min(j,q)\le k<\max(j,q)} a_{ij}\cdot a_{pq} \le c_g(k) \\
& \forall g\in G, \forall k \in (1,L) && \text{(7d)}\\
& a_{ij} \text{ is binary.} && \text{(7e)}
\end{aligned}$$

Constraint (7b) is to ensure that each segment of nets would be assigned to one and only one layer. Each edge $e \in E$ is associated with one capacity $c_e(i,j)$, and constraint (7c) is for the edge capacity of each layer. Constraint (7d) is for the via capacity in each layer, which restricts the available via capacity for each layer at certain grid position.

First, we show that if each $C_{\text{down}}(s_i)$ is constant, the TILA can be formulated as an ILP, then a mature ILP solver is possible to be applied. Here, $C_{\text{down}}(s_i)$ is downstream capacitance of segment s_i. We can use a Boolean variable $\gamma_{ij,pq}$ to replace each nonlinear term $a_{ij} \cdot a_{pq}$. Then, (7) can be transferred into ILP through introducing the following artificial constraints:

$$\begin{cases} a_{ij} + a_{pq} \leq \gamma_{ij,pq} + 1 \\ a_{ij} \geq \gamma_{ij,pq}, a_{pq} \geq \gamma_{ij,pq}. \end{cases} \tag{8}$$

Due to the computational complexity, ILP formulation suffers from serious runtime overhead, especially for those practical routing test cases. A popular speedup technique is to relax the ILP into linear programming (LP) by removing the constraint (7e). It is obvious that the LP solution provides a lower bound to the original ILP formulation. We observe that the LP solution would be like this: each a_{ij} is assigned to 0.5 and each $\gamma_{ij,pq}$ is 0. By this way, all the constraints are satisfied, and the objective function is minimized. However, all these 0.5 values to a_{ij} provide no useful information in guiding the layer assignment, as we prefer each a_{ij} closes to either 0 or 1. In other words, the LP relaxation is hard to provide reasonable good solution. Instead of expensive ILP formulation or its LP relaxation, our framework proposes a Lagrangian relaxation-based algorithm to solve the original (7).

B. Lagrangian Relaxation-Based Optimization

Lagrangian relaxation [38] is a technique solving optimization problems with difficult constraints, where some or all hard constraints are moved into objective function. In the updated objective function, each new term is multiplied with a constant known as Lagrange multiplier (LM). Our idea is to relax the via capacity constraint (7d) and incorporate it into the objective function. We specify each $a_{ij} \cdot a_{pq}$ a non-negative LM $\lambda_{ij,pq}$, and move the constraint into objective function. The modified formula is called LRS, as shown in (9). Through this relaxation methodology, via capacity overflow is handled with timing optimization simultaneously

$$\begin{aligned} \min \quad & \sum_{i \in S} \sum_{j=1}^{L} d_e(i,j) \cdot a_{ij} \\ & + \sum_{(i,p) \in E_x} \sum_{j=1}^{L} \sum_{q=1}^{L} \sum_{k=\min(j,q)}^{\max(j,q)-1} d_v(i,p,k) \cdot a_{ij} \cdot a_{pq} \\ & + \sum_{(i,p) \in E_x} \lambda_{ij,pq}(a_{ij} \cdot a_{pq} - c_g(k)) \\ \text{s.t.} \quad & \text{(7b), (7c), (7e).} \end{aligned} \tag{9}$$

It is known that for any fixed set of LM $\lambda_{ij,pq}$, the optimal result to the LRS problem is smaller or equal to the optimal solution of the original (7) [38]. That is, the original formulation is the primal problem and the LM optimization is the dual problem. Therefore, the Lagrangian dual problem (LDP) is to maximize the minimum value obtained for the LRS problem by updating LMs accordingly.

Algorithm 1 gives a high level description of our Lagrangian relaxation-based framework to the TILA problem. The inputs are an initial layer assignment solution and a critical net ratio value α. Based on the α value we select some critical nets and noncritical nets (line 1). All the segments belonging to these (selected critical and noncritical) nets are reassigned layers by our incremental framework. Please refer to Section III-D for more details of our critical and noncritical net selection.

Algorithm 1 TILA

```
Require: Initial layer assignment solution;
Require: Critical net ratio $\alpha$;
1: Select all segments based on $\alpha$;                ▷ Section III-D
2: Initialize $C_{down}(s_i)$ for each segment $s_i$;
3: Initialize LMs;
4: while not converged do
5:     Solve LRS;                                          ▷ Section III-C
6:     Update $C_{down}(s_i)$ for all $s_i$;
7:     Update LMs;
8: end while
```

Based on the initial layer assignment solution, we initialize all the $C_{\text{down}}(s_i)$ for each selected segment s_i (line 2). The LMs are also initialized in line 3. In our implementation, the initial values of all LMs are set to 2000. Our framework iteratively solves a set of LRS, with fixed LM values (lines 4–8). In solving LRS, we minimize the objective function in (9) based on the current set of LMs. The details of solving LRS are discussed in Section III-C. After solving each LRS, we recalculate the downstream capacitances of all the segments $C_{\text{down}}(s_i)$ based on (1) (line 6). We use a subgradient-based algorithm [39] to update the LMs to maximize LDP (line 7). In more details, the LM in the current iteration is dependent on the LM from the last iteration $\lambda'_{i,j,p,q}$, the step length θ_{ijpq}, and the available resources

$$\lambda_{i,j,p,q} = \lambda'_{i,j,p,q} + \theta_{ijpq} \cdot \left(a_{ij} \cdot a_{pq} - c_g\right). \tag{10}$$

The available via resources can be obtained directly by updating the current via capacity as in [35]. To decide the step length, we adopt the classic calculation as follows:

$$\theta_{ijpq} = \frac{\phi \cdot \left[\text{UB} - L\left(\lambda_{i,j,p,q}\right)\right]}{\left\|\left(a_{ij} \cdot a_{pq} - c_g\right)\right\|^2}. \tag{11}$$

Based on (11), UB refers to the upper bound of the total costs of via v and segments connecting to v, while $L(\lambda_{i,j,p,q})$ refers to the current total costs. ϕ is the scaling factor traditionally from 2 to 0, and here we choose it as 1 for convenience. Through this updating procedure, LMs help to fix the potential via violations. In our implementation, the iteration in line 4 will end if one of the following two conditions is satisfied: either the iteration number is larger than 20; or both the wire delay improvement and the via delay improvement are less than a prespecified fraction.

C. Solving Lagrangian Subproblem

Through removing the constant items and reorganizing objective function of (9), we rewrite LRS into

$$\begin{aligned} \min \quad & \sum_{i=1}^{S} \sum_{j=1}^{L} c(i,j) \cdot a_{i,j} + \sum_{(i,p) \in E_x} \sum_{j=1}^{L} \sum_{q=1}^{L} c(i,j,p,q) \cdot a_{ij} \cdot a_{pq} \\ \text{s.t.} \quad & \text{(7b), (7c), (7e)} \end{aligned} \tag{12}$$

where

$$\begin{cases} c(i,j) & = d_e(i,j) \\ c(i,j,p,q) & = \sum_{k=\min(j,q)}^{\max(j,q)-1} d_v(i,p,k) + \lambda_{ij,pq}. \end{cases}$$

Theorem 1: For a set of fixed $\lambda_{ij,pq}$, LRS is $\mathcal{NP}$-hard.

Due to space limit, the detailed proof is omitted. Because of nonlinear term $a_{ij} \cdot a_{pq}$, the proof can be through a reduction from quadratic assignment problem [40]. In addition,

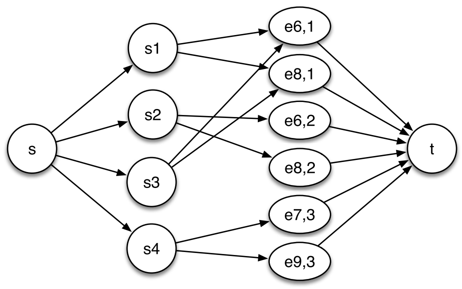


Fig. 7. Example of min-cost flow model.

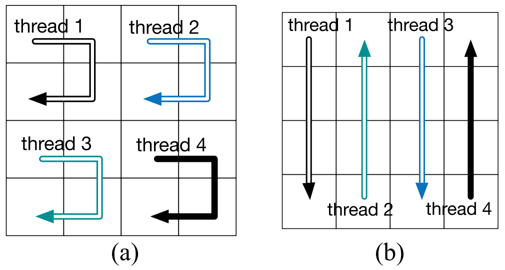


Fig. 8. Our parallel scheme to support multithreading computing on $K \times K$ partitions. (Here $K = 4$). (a) Parallel pattern 1. (b) Parallel pattern 2.

unless $\mathcal{P} = \mathcal{NP}$, the quadratic assignment problem cannot be approximated in polynomial time within some finite approximation ratio [41]. Inspired by McCormick envelops, we prefer to linearize the term $a_{ij} \cdot a_{pq}$

$$c(i, j, p, q) \cdot a_{ij} \cdot a_{pq} \approx c(i, j, p, q) \cdot \left(a'_{pq} \cdot a_{ij} + a'_{ij} \cdot a_{pq}\right) \quad (13)$$

where a'_{pq} is the value of a_{pq} in previous iteration, and a'_{ij} is the value of a_{ij} in previous iteration. This linearization is based on the segment assignment of the last iteration. Since LRS is solved iteratively through updating LMs, this approximation is acceptable. Taking $a_{18} \cdot a_{29}$ as an instance, where a'_{18}, a'_{29} are 1, we can obtain that segments s_1 and s_2 are assigned on layers 8 and 9 in the previous iteration, respectively. This means that segments s_1 and s_2 should belong to critical nets because they have been assigned on high metal layers by our framework. Thus, in later iterations, when considering the assignment of segment s_1, we assume that segment s_2 is assigned on layer 9, and vice versa, according to (13). In this manner, segments s_1 and s_2 are probable to be assigned on high metal layers as before. Since each critical segment has a tendency to be assigned on high metal layers, the problem converges after several iterations.

Through the linearization technique in (13), the objective function in (12) is a weighted sum of all the a_{ij}. We will show that the linearized LRS can be solved through a min-cost network flow model. The basic idea is that the weighted sum of all the a_{ij} can be viewed as several assignments from segments to layers, while the weight of each a_{ij} is the cost to assign segment i to layer j. Constraints (7b) and (7c) can be integrated into the flow model through specified edge capacity. Constraint (7e) is satisfied due to the inherent uni-modular property of min-cost network flow [39].

An example of such min-cost flow model is illustrated in Fig. 7. Given four different segments s_1, s_2, s_3, s_4 and several edges, we build up a directed graph $G = (V, E)$ to represent the layer assignment relationships. The vertex set V includes four parts: 1) start vertex s; 2) segment vertices V_S; 3) layer vertices V_L; and 4) end vertex t. Here, both start and end vertices are pseudo vertices. Segment vertices V_S represent a collection of segments to be assigned, where the collection size is equal to the number of segments. Similarly, a layer vertex in V_L represents a layer on which a segment can be reassigned. The edge set E is composed of three sets of edges: 1) $\{s \rightarrow V_S\}$; 2) $\{V_S \rightarrow V_L\}$; and 3) $\{V_L \rightarrow t\}$. Notably, here the edge set E represents the edges in the network flow, while the layer vertices represent the layers of edges in the global routing grid model. We define all the edge costs as follows: the cost of one edge from V_S to V_L is the cost of assigning the segment to corresponding layer; the costs of all other edges are set to 0. We define all the edge capacities as follows: the capacity of one edge from V_L to node t is the capacity of the corresponding edge in the routing grid model; while the capacities of all other edges are set to 1. Then edge capacity constraint can be satisfied by the capacity of edge from V_L to node t, and the capacity from node s to V_S guarantees that one segment can just be assigned on one layer. As shown in Fig. 7, segment s_1 can be assigned on either layer 6 or 8 of edge 1. The numbers shown in V_L vertices indicate the specified layer of this edge and the edge index, respectively. The corresponding nets are given in Fig. 5, where we can see that segment s_1 shares one edge with s_3 for resource competition. Meanwhile, segment s_4 has a different routing direction with the other segments so it has to be assigned on other layers. In this example, we assume that each segment passes through one edge with its length equal to the grid size, as shown in Fig. 5. For a segment passing through multiple edges, we prefer to split it into a set of subsegments, and each subsegment has the same length as the grid size. We construct the flow graph, where each subsegment has its own assigning cost, and the number of subsegments to be assigned on one layer is also constrained by the layer node.

D. Critical and Noncritical Net Selection

Given an input ratio value α, our framework would automatically identify $\alpha\%$ of the total nets as critical nets, while other $\alpha\%$ of the total nets as noncritical nets. Both the selected critical nets and the selected noncritical nets would be reassigned layers. The motivation of critical net selection is to reassign their layers to improve timing, while the motivation of noncritical net selection is to release some high layer resources to the critical nets. By this way, our flow is able to overcome the limitation of any net order.

To identify all the critical nets can be trivial: first all the net timing costs in original layer assignment are calculated, and then the $\alpha\%$ of worst delays are selected. Yet, noncritical net selection is not so straightforward, as randomly selecting $\alpha\%$ of best timing nets may not be beneficial to critical net timing. In our implementation, we check the $2 \cdot \alpha$ best timing nets to associate each net with a score to indicate their overlapping resources with critical nets. Meanwhile, if there is an overlap with critical nets, the assigned layer of this short net should be higher than the lowest layer of these critical nets. Otherwise, it is not regarded as an effective overlap. Then we select half of them with the best scores as noncritical nets.

E. Parallel Scheme

Our framework supports parallel scheme by dividing the global routing graph into $K \times K$ parts. An example of such division is illustrated in Fig. 8(a), where each division is solved separately and $K = 4$. To ensure each segment to be solved in

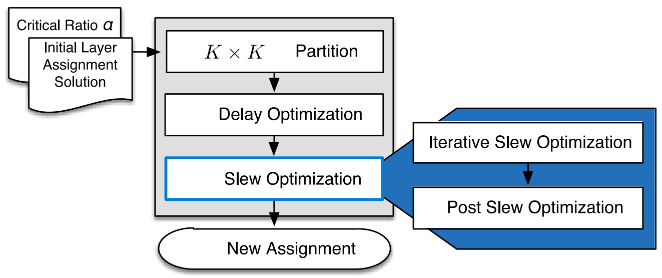


Fig. 9. Overall timing optimization flow.

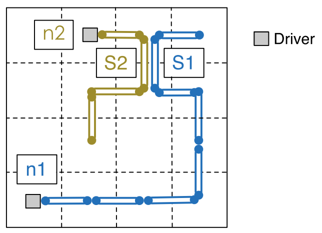


Fig. 10. Example of difference between delay and slew optimization.

TABLE II
NOTATIONS USED FOR SLEW MODEL

N_{slw}	set of nets with slew violations
$P_{critical}$	path with slew violations
$p_d(s_i)$	downstream node of segment s_i
$p_u(s_i)$	upstream node of segment s_i
$Slw_{sink}(P_{critical})$	sink slew of critical path $P_{critical}$
$Slw(p_d(s_i))$	output slew of segment s_i
$Slw(p_u(s_i))$	input slew of segment s_i
$Slw_{step}(i,j)$	step slew of segment s_i on layer j
$Slw_e(i,j)$	output slew of segment s_i assigned on layer j
Slw_c	given slew constraint
Slw_{imp}	most slew improvement
$\delta Slw(i,l)$	slew improvement by assigning s_i on layer l
δSlw_{ip}	slew improvement by switching s_i and s_p

only one partition, for those crossing boundaries of partitions, they are assigned in the same partition as its geometric center. The reason of such division is twofold. First, our Lagrangian relaxation-based optimization is to solve a set of min-cost flow models, as discussed in Sections III-B and III-C. The runtime complexity to solve a single flow model is $\mathcal{O}(|V| \cdot |E|)$, where $|V|$ and $|E|$ are the vertex number and the edge number of the graph. Dividing the whole problem into a set of subproblems can achieve significant speed-up. In addition, multithreading is applied to provide further speed-up. Second, inspired by the Gauss–Seidel method [42], when one thread is solving flow model in one partition, the most recently updated results by peer threads are taken into account, even if the updating occurs in the current iteration. Besides, we propose a more general type of parallel pattern suitable for any $K \times K$ partition, as in Fig. 8(b). In this example, neighboring threads start in inverse directions and avoid operating on neighboring partitions simultaneously as much as possible. After solving different partitions, we synchronize the newly updated layer assignment results to eliminate the potential conflicts.

F. Iterative Slew Optimization

During timing closure, slew violations are important performance metrics that may cause a huge demand for buffering resources. Thus, we should also focus on reducing slew violations besides delay optimization. Fig. 9 depicts the overall algorithm flow, which mainly consists of two stages: 1) delay optimization and 2) slew optimization. The details of delay optimization are already introduced from Sections III-B to III-E. As discussed in Section II-C, segment step slew is in proportion to its delay. With the constant input slew, the higher layer this segment is assigned, the fewer output slew can be obtained. Therefore, delay optimization is deemed to mitigate slew violations. Nevertheless, segment delay optimization mainly considers the layer assignments of its downstream segments due to the existence of downstream capacitance, but neglects its upstream segments. Since layer assignments of the upstream segments affect the segment input slew, the upstream segments should also be taken into accounts.

An example is given in Fig. 10. Here, we assume that both net n_1 and net n_2 are critical while there is only one available routing capacity for each edge, so segments s_1 and s_2 should compete for the layer resources. Regarding delay optimization, segment s_2 is likely to be assigned on a higher layer because it owes a larger downstream capacitance; while in fact, segment s_1 should be placed on a higher layer because a longer path may introduce slew violations. Thus, after our slew optimization flow, segment s_1 will be assigned a higher priority on a higher layer. The main reason is that slew optimization considers both upstream segments and downstream segments. In this manner, slew optimization has a different impact on assignment of critical nets in comparison to delay optimization. The detailed reasons are twofold: first, critical nets can be selected in a different way during delay and slew optimization. In the stage of slew improvement, these nets exceeding slew constraints are to be selected; however, in the first stage we mark these nets with higher total delays as critical nets. This may induce potential discrepancies for nets to be optimized. Second, delay improvement targets at total delay reduction, while slew improvement targets at slew violations reduction. Due to different optimal objectives, assigning costs for both delay and slew optimization may lead to a tradeoff based on their weights. Considering the assigning differences of s_1 and s_2 in Fig. 10, possible oscillation may be introduced by setting different weights to delay and slew optimization. Therefore, due to the differences of selected nets and optimal objectives, we prefer to target delay and slew separately, and reduce slew violations globally as a second stage.

Fig. 9 also outlines the slew optimization flow, whose input is the assignment result after delay optimization. The slew optimization consists of two steps: 1) iterative slew optimization and 2) post greedy optimization. This section focuses on the first step to reduce slew violations based on flow model, while Section III-G provides the details of post slew optimization. Some notations used in slew optimization are listed in Table II. In the iterative flow, a part of critical and noncritical nets are selected for optimization. To calculate the net criticality, we divide the net into a set of paths, and calculate the sink slew of each path. If the sink slew exceeds the slew constraint, this path is defined as a critical path, i.e., P_{critical}, and the exceptional slew is counted as critical value. Meanwhile, segment input slews are initialized based on the input result and then we reassign these nets through the iteration-based optimization. When the number of slew violations converges to a certain ratio, the iteration-based optimization stops.

Now, we go over the details about how to solve the problem through min-cost flow model. First all the segments on critical paths are considered because their layer assignments affect the path sink slew. During slew optimization, we lower the slew constraint by 5% in order to leave enough slew slacks. Equation (14) gives the slew constraint

$$\mathrm{Slw}(p_d(s_i)) \leq 0.95 \cdot \mathrm{Slw}_c, \qquad i \in P_{\mathrm{critical}} \tag{14}$$

where $\mathrm{Slw}(p_d(s_i))$ is the segment output slew, and Slw_c is the slew constraint. To solve this problem, we relax (14) through Lagrangian relaxation by moving the slew calculation into the objective function, and eliminate all the $0.95 \cdot \mathrm{Slw}_c$ because they are constants. Equation (15) provides the slew optimization formulation, where each segment slew is multiplied with a Lagrangian multiplier (LM), i.e., β_{ij}, which is set to 1 as the initial value

$$\min \sum_{i \in P_{\mathrm{critical}}} \sum_{j=1}^{L} \beta_{ij} \cdot \mathrm{Slw}_e(i,j) \cdot a_{ij}$$
$$\text{s.t.} \quad (7b)-(7e). \tag{15}$$

During each iteration, LMs are updated as

$$\beta_{ij} = \beta'_{ij} \cdot \sqrt{\frac{\mathrm{Slw}_{\mathrm{sink}}(P_{\mathrm{critical}})}{\mathrm{Slw}_c}} \tag{16}$$

where β'_{ij} is the LM in previous iteration, and $\mathrm{Slw}_{\mathrm{sink}}(P_{\mathrm{critical}})$ is the sink slew of critical path P_{critical}. With the consideration of sink slew, we impose more weights on longer paths. Therefore, in the example of Fig. 10, segment s_1 has a higher priority than s_2.

Similar with (7), (15) is solvable through ILP because we can obtain $\mathrm{Slw}_e(i,j)$ based on the last iteration. Still, we incorporate the via capacity constraints into the objective function with the same linearization method as in (13). Ultimately, the problem can be formulated as a weighted sum of a_{ij}s and solved through min-cost max-flow model. After solving the problem in each iteration, we update the input slews and check if there is a convergence of slew violations. If the improvement is below a certain ratio, then this flow terminates. In summary, this algorithm provides a slew targeted optimization because it considers both upstream and downstream segments. Meanwhile, more emphasis is placed on critical paths by taking sink slews into accounts.

Based on the slew model, segment input slew affects output slew directly, but during each iteration, we obtain the input slew based on the last iteration, which may introduce slew discrepancies. Therefore, we implement a post slew optimization algorithm, which focuses on fixing local violations while considering current layer assignments of the whole path. The details of this algorithm are given in Section III-G.

G. Post Slew Optimization

In this section, we propose a post slew optimization algorithm to further reduce slew violations. The pseudo code is shown in Algorithm 2. Based on the current results, we traverse each net to check if there exist slew violations. For those nets with violations, they are saved in a net set, i.e., N_{slw}, and sorted in the descending order of slew violations (line 2). The net with the highest priority is the one with the most segments causing slew violations. To cope with slew violations, we start from the first segment on the critical path (line 4), and adjust the layer assignment of each segment s_i through two steps (lines 5–31).

First, if there is any available routing capacity for s_i on higher layers (line 7) and its segment slew can be improved (line 8), we record the improvement and mark this layer as a candidate (line 9). Meanwhile, the induced via capacity violations cannot exceed a given ratio, Ra. After traversing each possible layer, the layer with the most improvement is selected for s_i to assign (line 13). In this way, the sink slews of other nets are not affected while the current segment output slew is improved. However, if no available layer is found, a second step is required (lines 14–28).

In the second step, we search for a noncritical segment on the same edge with s_i. When exchanging its layer with segment s_i, we would not degrade its slew much while improving the output slew of s_i. In order to find this segment, we traverse each noncritical segment s_p that is assigned on a layer higher than $l(s_i)$ and able to bring slew improvements for s_i (lines 16–18). Then the slew improvement is calculated by switching the layers of segment s_i and segment s_p (line 19). If the improvement outperforms the current improvement, we signify this segment as s_{temp}, and record its layer (lines 20–24). Here, we also take into accounts the net which segment s_p belongs to. When its sink slew is close to the given slew constraint, then segment s_p will not be considered as an exchange candidate. After traversing each segment on higher layers, we switch the assigned layers of segments s_i and s_{temp} and update the slews of their nets (lines 26 and 27). When the slew violation of P_{critical} has been fixed, then we continue to fix the next net in N_{slw}. Besides, if a segment has already exceeded the slew constraint, we will skip the remaining segments in this net because there is no further optimization space for sink slews of this net. By this way we can save the runtime overhead

Algorithm 2 Post Slew Optimization Algorithm

```
Require: Current layer assignment solution;
1: Save all slew critical nets in $N_slw$;
2: Sort nets in the descending order of slew violations;
3: for each net $n \in N_slw$ do
4:     for each $s_i \in P_{critical}$ do
5:         Initialize $Slw_{imp} = 0$;
6:         for each $l \in e(s_i)$ do
7:             if Routing capacity exists for layer $l$ then
8:                 if $\delta Slw(i, l) \geq Slw_{imp}$ and OV $\leq$ Ra then
9:                     Update $l_{temp}$ and $Slw_{imp}$;
10:                end if
11:            end if
12:        end for
13:        Assign $s_i$ on $l_{temp}$;
14:        if No $l_{temp}$ is found then
15:            for each noncritical $s_p$ on $e(s_i)$ do
16:                if $\delta Slw(i, l(s_p)) \leq 0$ then
17:                    Continue;
18:                end if
19:                $\delta Slw_{ip} = \delta Slw(i, l(s_p)) + \delta Slw(p, l(s_i))$;
20:                if $\delta Slw_{ip} \geq Slw_{imp}$ and OV $\leq$ Ra then
21:                    if $Slw_{n(s_p)} \leq \alpha \cdot Slw_c$ then
22:                        Update $s_{temp}$ and $Slw_{imp}$;
23:                    end if
24:                end if
25:            end for
26:            Switch layers between $s_i$ and $s_{temp}$;
27:            Update $Slw$ for $n(s_i)$ and $n(s_{temp})$;
28:        end if
29:        if $Slw_{sink}(P_{critical}) \leq Slw_c$ or $Slw(i, l') \geq Slw_c$ then
30:            break;
31:        end if
32:    end for
33: end for
```

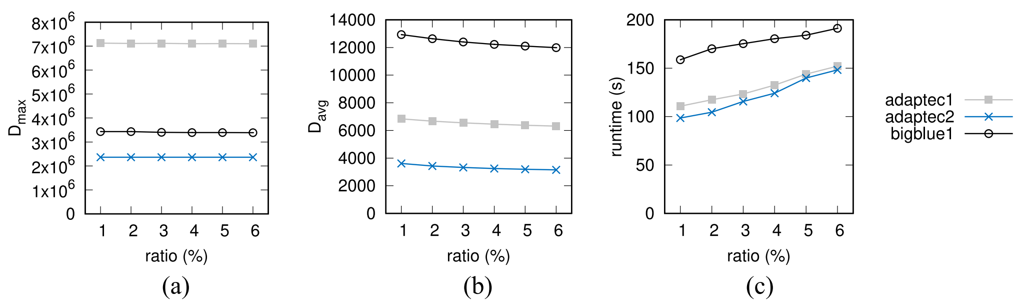


Fig. 11. Performance impact on different ratio values. Impact of ratio on (a) maximum delay, (b) average delay, and (c) runtime.

TABLE III
NORMALIZED CAPACITANCE AND RESISTANCE

Wire [11]			Via	
Layer	C	R	Layer	R
M1	1.14	23.26	$v_{1,2}$	25.9
M2	1.05	19.30	$v_{2,3}$	16.7
M3	1.05	23.26	$v_{3,4}$	16.7
M4	0.95	5.58	$v_{4,5}$	16.7
M5	1.05	3.26	$v_{5,6}$	5.9
M6	1.05	3.26	$v_{6,7}$	5.9
M7	1.05	3.26	$v_{7,8}$	5.9
M8	1.00	3.26	$v_{8,9}$	1.0
M9	1.05	1.00	$v_{9,10}$	1.0
M10	1.00	1.00	-	-

(lines 29–31). The algorithm ends until all nets in N_{slw} are traversed. In comparison to slew optimization in Section III-F, this algorithm adjusts the layer assignment of segments based on their real input slew, thus providing a more accurate slew optimization. Meanwhile, if there are only a few slew critical nets, it is efficient to fix the violations through this algorithm.

IV. Experimental Results

We implemented the proposed framework in C++, and tested it on a Linux machine with 2.9 GHz Intel Core and 192 GB memory. We selected open source graph library LEMON [43] as our min-cost network flow solver, and utilized OpenMP [44] for parallel computing. In our implementation, the default K is set to 6, and the default thread number is set to 6.

A. Evaluation on ISPD 2008 Benchmarks

In the first experiment, we evaluate our framework on ISPD 2008 benchmarks [45]. The NCTU-GR 2.0 [9] is utilized to generate the initial global routing solutions. The initial layer assignment results are from negotiation-based via minimization (NVM) [18], which is targeting at via number and overflow minimization. Our framework is tested the effectiveness to incrementally optimize the timing. To calculate the wire delay in (1) and via delay in (2), all the metal wire resistances, metal wire capacitances, and via resistances are listed in Table III. Column "C" lists the capacitance. Columns "R" list the resistances for wire layers and via layers, respectively. The resistances and capacitances of wires are directly from [11], while the via resistance values are normalized from industry settings in advanced technology nodes. Since ISPD 2008 benchmarks do not provide the input capacitance and output resistance values of sinks, here we assume they are zero.

Table IV compares NVM [18] with our incremental layer assignment tools TILA-1% and TILA-5%. NVM provides a minimum number of vias during layer assignment with very low runtime overhead. In "TILA-1%" and "TILA-5%" the ratio value α are set to 1% and 5%, respectively. That is, in TILA-1%, 1% of timing critical nets and 1% of noncritical nets are reassigned layers. In TILA-5%, 5% of timing critical nets and 5% of noncritical nets are reassigned layers. For each methodology, columns "OE#," "OV#," "D_{avg}," "D_{max}," and "via#" list the resulting edge overflow, via overflow number, average delay, maximum delay, and total via number, separately. Here, the calculation of via overflow is described in [35]. Besides, "CPU(s)" reports the runtime in seconds for both NVM and TILA. We do not test our tools on test case `newblue3` as NCTU-GR [9] cannot generate a legal global routing solution, where the number of segments passing one edge in 2-D exceeds the total edge capacities. We also cannot report the results from another recent work [19], as for this benchmark suite their binary gets assertion fault before dumping out results.

From Table IV we can see that in TILA-1%, when 1% of the most critical nets are shuffled layers, maximum delay can be reduced by 53% on the ISPD 2008 benchmarks. Meanwhile, the overflow number and the average delay are reduced by 3% and 10%, respectively. The penalty of such timing improvement is that the via number is increased by only 3%. On the average, TILA-1% requires around 409 s for each test case. Compared with fast net-by-net solver NVM, although our planner solves a global optimization problem, its runtimes are reasonable. In TILA-5%, when 5% of the most critical nets are reassigned layers, the maximum delay is reduced by 53%. Meanwhile, the overflow number and the average delay are reduced by 3% and 19%, respectively. The penalty of TILA-5% is that the via number increases by 11%. From Table IV we can see that even small amount of critical nets (e.g., 1%) are considered, the maximum delay can be effectively optimized. When more nets are inputted, better average delay and less overflow number are expected. We pay a penalty of increasing via counts to achieve better timing results with more released nets. Meanwhile, runtime shows a slight increase with more reassigned nets because of the larger problem size. In addition, our framework is with good scalability, i.e., with problem size increases fivefold, the runtime of TILA-5% is just around one and half times of TILA-1%.

Critical net ratio α is a user-defined parameter to control how many nets are released to incremental layer assignment. In Table IV, ratio α is set to 1% and 5%. Fig. 11 analyzes the impact of ratio value to the performance of incremental layer assignment framework. Fig. 11(a) shows the impact on the maximum delay, where we can see that the maximum delays are kept the same. This means for these test cases, releasing

TABLE IV
PERFORMANCE COMPARISONS ON ISPD 2008 BENCHMARKS

Bench	NVM [18]						TILA-1%						TILA-5%				
	OE#	OV#	D_{avg} (10^3)	D_{max} (10^3)	via# (10^5)	CPU (s)	OE#	OV#	D_{avg} (10^3)	D_{max} (10^3)	via# (10^5)	CPU (s)	OV#	D_{avg} (10^3)	D_{max} (10^3)	via# (10^5)	CPU (s)
`adaptec1`	0	48588	7.26	8776.6	19.03	36.2	0	50716	6.84	7126.0	19.26	124.6	53472	6.37	7107.2	20.18	146.6
`adaptec2`	0	39468	4.35	14424.9	19.01	31.5	0	36824	3.61	2365.8	19.38	115.6	32266	3.19	2365.8	20.63	145.3
`adaptec3`	0	91996	9.70	24998.9	36.29	89.3	0	89800	8.67	7861.3	36.77	396.5	89598	7.89	7860.0	38.83	796.3
`adaptec4`	0	77542	6.96	38646.7	31.56	55.1	0	67946	5.89	9745.2	32.55	330.7	56037	5.25	9746.0	34.80	562.5
`adaptec5`	0	79101	10.95	9958.0	54.30	98.5	0	81956	9.98	8740.2	55.43	493.4	85590	9.11	8693.1	58.54	587.2
`bigblue1`	0	43029	13.50	3675.4	21.25	48.4	0	46151	12.93	3434.7	21.68	235.7	52779	12.10	3390.4	22.67	246.9
`bigblue2`	12	117989	3.02	58259.1	42.70	48.8	12	114215	2.63	18294.9	43.44	208.4	114220	2.44	18279.0	45.35	239.3
`bigblue3`	0	66790	4.98	3122.2	51.29	81.4	0	65437	4.15	2708.9	53.22	378.4	66639	3.49	2710.1	60.04	675.6
`bigblue4`	447	97355	8.22	53401.4	107.65	169.4	447	114215	7.08	35310.7	111.01	743.6	113744	6.08	35320.1	122.08	984.4
`newblue1`	179	58656	1.21	670.7	22.03	21.6	179	56602	1.00	566.2	22.39	99.1	51721	0.93	565.4	23.67	122.8
`newblue2`	0	40959	4.31	12265.2	28.36	35.3	0	33941	3.97	10569.2	29.02	159.2	19997	3.57	10567.1	31.04	253.3
`newblue4`	108	88220	4.17	15478.3	46.85	83.2	108	84273	3.88	8976.9	47.65	302.7	77931	3.55	8963.8	50.41	429.5
`newblue5`	0	160141	6.19	11910.3	84.61	136.6	0	151300	5.64	4551.7	86.88	644.2	141974	5.12	4552.9	93.86	991.8
`newblue6`	0	94425	7.28	18987.0	77.43	103.4	0	96740	6.57	3963.7	78.67	686.8	105034	5.99	3964.6	82.39	842.6
`newblue7`	369	146737	7.01	13416.0	160.57	236.7	369	141936	5.91	12028.2	166.58	1213.3	158329	5.06	12033.0	183.94	1427.9
average	74	83400	6.61	19199.4	53.5	85.0	74	81121	5.92	9082.9	54.93	408.8	81289	5.34	9074.6	59.23	563.5
ratio	1.00	**1.00**	**1.00**	**1.00**	1.00	1.00	1.00	**0.97**	**0.90**	**0.47**	1.03	–	**0.97**	**0.81**	**0.47**	1.11	–

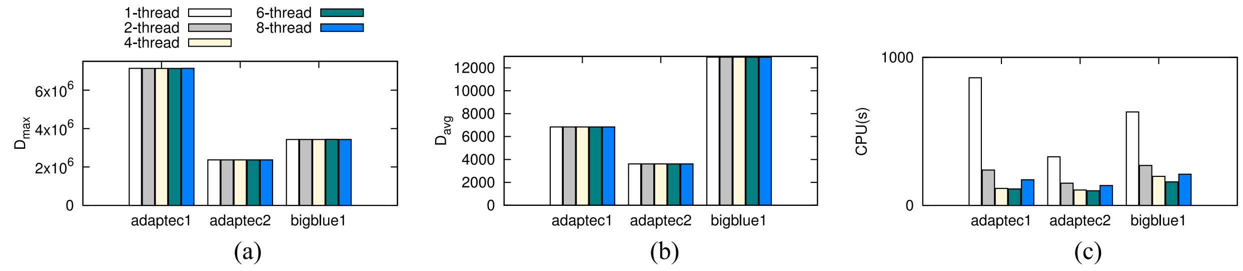


Fig. 12. Evaluation thread number impact on three test cases in ISPD 2008 benchmark suite. Impact on (a) maximum delay, (b) average delay, and (c) runtime.

1% of critical nets is enough for maximum delay optimization. Fig. 11(b) shows the impact on the average delay, where we can see increasing the ratio value can slightly improve the average delay. Fig. 11(c) is the impact on the runtime, where we can see that the runtime increases along with the increase of ratio value. From these figures we can see that the ratio value can provide a tradeoff between average delay and the speed of our tool.

Our incremental layer assignment utilizes OpenMP [44] to implement multithreading. Fig. 12 analyzes the performance of our framework under different partition and thread numbers. Thread 1 corresponds to 1×1 partition, thread 2 corresponds to 2×2 partitions, and so on. With more partitions, the size of network flow model is reduced quadratically thus benefiting the runtime significantly together with multithreads. From Fig. 12(a) and (b) we can see that the impact of thread number on both maximum delay and average delay is insignificant. From Fig. 12(c) we can observe that more thread number can achieve more speed-ups. However, when thread number is larger or equal to 6, the benefit to runtime is not clear. Therefore, in our implementation the thread number is set to 6.

To demonstrate the benefit of solving the problem in a global manner, we implement a greedy strategy to assign segments in a net-by-net manner. All the reassigned nets are sorted based on their timing priorities so that a more critical net has a higher priority for resources. For each net, segments are traversed sequentially and layers are selected based on the same costs as that in min-cost max-flow network. Here, we release 1% critical nets and 1% noncritical nets. The results are shown in Fig. 13. Observe that for both average and maximum delay TILA can achieve slightly better results compared with the greedy method. The main reason is that the greedy method assigns higher priorities to those critical nets so that they are able to utilize higher layer resources. Thus, significant timing optimization can also be achieved through this greedy methodology. Nevertheless, they sacrifice the via capacity violations due to their preferences to high layers. Regarding runtime, as shown in Fig. 13(d), due to the net-by-net scheme, the greedy method is faster than TILA. Therefore, to control timing optimization and capacity constraints in a reasonable manner, a global optimization engine is more promising.

B. Evaluation on 20 nm Industry Benchmarks

In the second experiment, we test our incremental layer assignment framework on eight 20 nm industry test cases (`Industry1`–`Industry8`). We called an industry tool to generate initial global routing and layer assignment solutions. Different from the preceding experiment, here we use industry resistance and capacitance values to calculate the wire delays and the via delays. Table V lists the details of performance evaluation, where for each method columns OV#, D_{avg}, D_{max}, and via# provide the overflow number, average delay, maximum delay, and total via number. Since the critical nets are provided in the benchmarks, the critical and noncritical selection phases are skipped in this benchmark suite. We can see that compared with industry layer assignment solution, our framework can achieve 60% maximum delay and 34% average delay improvement. The total via number is very similar to the initial solution, and the factors are as follows: first, critical segments are assigned on high metal layers while noncritical segments are assigned on low layers with their neighboring segments. Few vias will be induced for those connecting segments are on close layers. Second, via delays are also included in our mathematical formulation, which also helps to control the via counts. Finally, industrial benchmarks

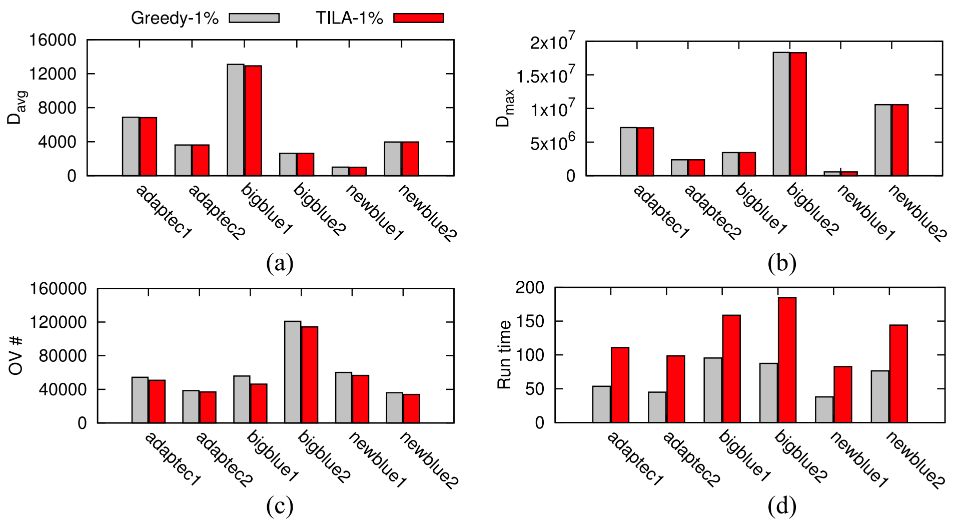


Fig. 13. Comparison between greedy methodology and TILA on some small test cases. (a) On average delay. (b) On maximum delay. (c) On via overflow. (d) On runtime.

TABLE V
PERFORMANCE COMPARISONS ON 20 nm INDUSTRY BENCHMARKS

Bench	Industry Layer Assignment				TILA				
	OV#	D_{avg}	D_{max}	via#	OV#	D_{avg}	D_{max}	via#	CPU(s)
`Industry1`	0	6204.0	68444.4	51805.0	0	3696.6	28667.2	49302.0	6.6
`Industry2`	0	6049.6	68713.0	52996.0	0	3796.4	27416.3	50331.0	7.0
`Industry3`	0	6025.4	81030.3	53905.0	0	3906.2	38230.8	51726.0	8.0
`Industry4`	0	5702.8	58478.5	56393.0	0	3669.2	25858.9	54188.0	9.3
`Industry5`	0	5531.4	78391.4	58944.0	0	3799.3	34347.0	56623.0	11.5
`Industry6`	0	5443.5	77803.0	60083.0	0	3692.9	33096.3	57456.0	12.7
`Industry7`	0	5066.0	114597.7	70658.0	0	3693.7	29348.7	70106.0	38.5
`Industry8`	0	4096.4	46893.7	75790.0	0	3040.2	20137.7	78823.0	127.8
average	0	5514.9	74294.0	60071.6	0	3661.8	29637.9	58569.4	127.8
ratio	**0**	**1.00**	**1.00**	**1.00**	**0**	**0.66**	**0.40**	**0.97**	-

provide an even assignment of segments through all the layers. This provides us potential spaces for via counts optimization. Besides, the initial solution is with zero overflow, and our framework can also maintain such zero overflow performance. In summary, from Table V we can see our incremental layer assignment framework can achieve significant timing improvement.

C. Slew Comparisons on ISPD and 20 nm Industry Benchmarks

In this section, we compare TILA with slew optimization (TILA-S) against TILA. Still, the effectiveness is verified by both ISPD and industry benchmarks with slew constraints. For ISPD benchmarks, the problem sizes are so different that one single constraint is not applicable to all benchmarks. Thus, we set the slew constraint of each benchmark as five times its initial average delay as shown in Table IV. In this way, the initial number of slew violations is in proportion to the size of each benchmark. However, the slew constraints for industry benchmarks are given based on industrial settings.

Table VI lists the results for ISPD benchmarks by comparing TILA-S-1% with TILA-1% while releasing 1%. Besides the performance metrics shown in Table IV, we introduce an additional column "SV#" which gives the number of slew violations, and the second column lists the initial number of violations. TILA-1% provides the intermediate results after delay optimization, while TILA-S-1% shows the final results. We can see that TILA-1% is able to reduce the slew violations significantly from 6.89×10^4 to 3.57×10^4, because delay optimization also benefits slew violations considering the downstream segments. However, with the slew targeted optimization, this number can further be reduced by 48%. Meanwhile, average delay also decreases by 2%, which shows that slew optimization can also benefit delay slightly. The maximum delay keeps similar with TILA, because its optimization space is limited after delay optimization. For vias and violations, there is no obvious difference between TILA-S and TILA. The main penalty of TILA-S is the 69% increase of runtime due to additional two-stage slew optimization. Based on the results, we observe that TILA-S can handle slew violations efficiently while keeping similar delay and via performance.

Fig. 14 shows the effect of adopting post slew optimization for some small cases of ISPD 2008 benchmarks. It is shown that the post slew optimization stage improves the number of slew violations slightly without affecting average delay and maximum delay. The main reason is that during selection of switching candidate segments, we take its current slew into consideration. Once the candidate is selected with the smallest slew degradation, its impact on delay is also negligible because slew is closely related with delay.

TABLE VI
COMPARISONS ON ISPD 2008 BENCHMARKS FOR SLEW OPTIMIZATION

Bench	NVM [18]	TILA-1%						TILA-S-1%					
	SV# (10^3)	SV# (10^3)	VO#	D_{avg} (10^3)	D_{max} (10^3)	via# (10^5)	CPU (s)	SV# (10^3)	VO#	D_{avg} (10^3)	D_{max} (10^3)	via# (10^5)	CPU (s)
adaptec1	8.57	4.59	50716	6.84	7126.0	19.26	110.8	3.76	50873	6.80	7128.8	19.30	185.5
adaptec2	24.75	10.38	36824	3.61	2365.8	19.38	98.6	6.22	36518	3.55	2365.9	19.53	158.7
adaptec3	19.77	8.22	89800	8.67	7861.3	36.77	361.2	7.09	89963	8.63	7861.4	36.88	614.9
adaptec4	54.23	16.05	67946	5.89	9745.2	32.55	330.7	12.28	67611	5.84	9744.9	32.66	510.6
adaptec5	54.65	21.35	81956	9.98	8740.2	55.43	493.4	14.32	83207	9.88	8724.2	55.70	869.3
bigblue1	16.68	8.12	46151	12.93	3434.7	21.68	158.8	6.21	46724	12.85	3438.8	21.75	407.0
bigblue2	81.77	59.00	114215	2.63	18294.9	43.44	184.7	43.59	113332	2.58	18299.9	43.77	437.1
bigblue3	67.42	38.06	65437	4.15	2708.9	53.22	378.4	19.86	63974	4.00	2710.2	54.33	732.4
bigblue4	118.28	67.48	98987	7.08	35310.7	111.01	743.6	28.50	98307	6.87	35414.9	113.11	1484.1
newblue1	46.67	36.60	56602	1.00	566.2	22.39	82.7	21.26	55417	0.98	566.1	22.78	132.6
newblue2	62.98	29.76	33941	3.97	10569.2	29.02	144.2	9.73	30043	3.85	10269.3	29.76	265.1
newblue4	52.56	25.43	84273	3.88	8976.9	47.65	302.7	12.42	83412	3.82	8973.8	48.14	396.4
newblue5	155.50	70.99	151300	5.64	4551.7	86.88	644.2	39.12	150477	5.53	4553.8	88.08	1169.0
newblue6	88.69	49.83	96740	6.57	3963.7	78.67	686.8	22.22	100305	6.39	3963.5	79.61	993.0
newblue7	181.17	89.48	141936	5.91	12028.2	166.58	1213.3	34.23	141209	5.71	12030.2	169.80	1695.7
average	68.91	35.69	81122	5.92	9082.9	54.93	408.8	18.68	80758	5.82	9069.7	55.68	670.1
ratio		**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**0.52**	**1.00**	**0.98**	**1.00**	**1.01**	1.69

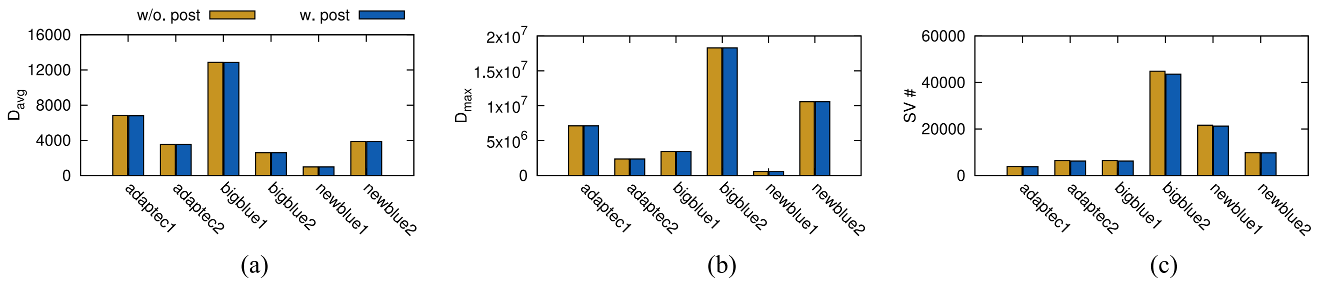


Fig. 14. Comparison between with and without post slew optimization stage on some small test cases. (a) On average delay. (b) On maximum delay. (c) On slew violations.

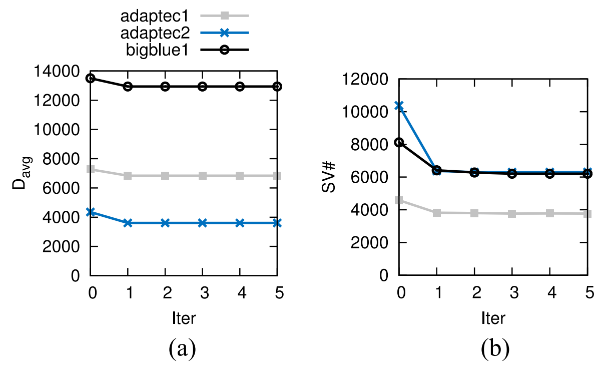


Fig. 15. Convergence with iteration number of TILA-S on some small test cases. (a) On average delay. (b) On slew violations.

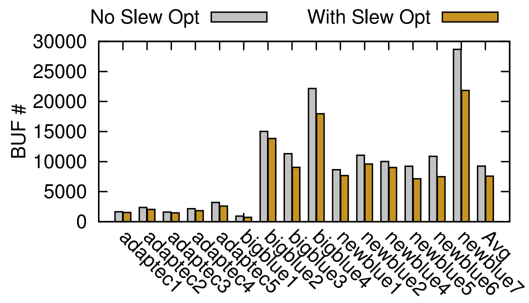


Fig. 16. Buffering overhead saving with slew optimization.

To illustrate the timing convergence of our iterative framework, we relax the convergence criteria for delay and slew optimization, and record the average delay and slew violations for each iteration till the fifth iteration in Fig. 15. The 0th iteration corresponds to the initial solution, where we can see a clear convergence after first two iterations.

As stated in Sections III-F and III-G, our slew optimization reduces slew violations and benefits buffering overheads. To show this effect, we measure the number of buffers we may adopt for ISPD benchmarks in Fig. 16. Here, we utilize a top-down algorithm to insert buffers in a net-by-net manner. For each net with slew violations, we traverse from its driver and insert a buffer when there is a slew violation; meanwhile, we assume the input slew of each net and the output slew from the buffer are both equal to 0. After traversing one net, we can obtain the number of buffers used in this net to fix the violations. It is shown that the average buffering cost can be reduced from 9258 to 7586 in Fig. 16. Therefore, our post slew-targeted optimization helps to reduce the buffering overhead, and is also able to provide an estimate of buffering costs at prebuffering stage.

For the 20 nm industry benchmarks, besides delay and via metrics, we also take slew violations into account. Table VII shows that the violations are reduced by 36%. This proves the efficiency of our slew optimization flow to fix some local violations. Meanwhile, since we target at improving the current segment slew without affecting others considerably, the average delay keeps the same as before. In addition, the maximum

TABLE VII
COMPARISONS ON 20 nm INDUSTRY BENCHMARKS FOR SLEW OPTIMIZATION

Bench	TILA					TILA-S				
	SV#	D_{avg}	D_{max}	via#	CPU(s)	SV#	D_{avg}	D_{max}	via#	CPU(s)
`Industry1`	24	3606.6	28667.2	49302	6.6	19	3686.2	27202.3	49308	7.1
`Industry2`	18	3796.4	27416.3	50331	7.0	14	3779.7	25934.5	50333	7.2
`Industry3`	10	3906.2	38230.8	51726	8.0	3	3898.8	34092.1	51742	8.5
`Industry4`	7	3669.2	25858.9	54188	9.3	2	3665.2	24159.9	54188	9.3
`Industry5`	0	3799.3	34347.0	56623	11.5	0	3799.3	34347.0	56623	11.5
`Industry6`	0	3692.9	33096.3	57456	12.7	0	3692.9	33096.3	57456	12.7
`Industry7`	0	3693.7	29348.7	70106	38.5	0	3693.7	29348.7	70106	38.5
`Industry8`	0	3040.2	20137.7	78823	127.8	0	3040.2	20137.7	78823	127.8
average	7.4	3650.6	29637.9	58569	27.7	4.8	3657.0	28539.8	58572	27.8
ratio	**1.0**	**1.00**	**1.00**	**1.00**	**1.00**	**0.64**	**1.00**	**0.96**	**1.00**	**1.01**

delay is reduced by 4%, because slew optimization considers the layer assignments of both upstream segments and downstream segments. We can also see that there is almost no difference for vias between TILA-S and TILA. Because of the very few number of slew violations in industrial benchmarks, we prefer to skip the first global optimal stage. The results from Table VII show the ability of post optimization stage to reduce violations with little runtime overhead. Therefore, with the additional slew optimization flow, TILA-S contributes lots of efforts to fixing slew violations while keeping similar delay performance as TILA, both for ISPD benchmarks and industrial benchmarks.

V. CONCLUSION

In this paper we have proposed a set of algorithms to the timing-driven incremental layer assignment problem while mitigating slew violations. At first the mathematical formulation is given to search for optimal total wire delays and via delays. Then Lagrangian relaxation-based method is proposed to iteratively improve the timing of all the nets. The LRS is modeled through min-cost flow model to provide effective integral solutions. In addition, multiprocessing of $K \times K$ partitions of the whole chip provides runtime speed up. Then we integrate the slew violation optimization method into our framework to mitigate the violations. Our incremental layer assignment tool with/without slew optimization, TILA-S, is verified in both ISPD 2008 and industry benchmark suites, and has demonstrated its effectiveness. In our current implementation, slew improvement is achieved through a separate stage with delay optimization. As a future work, we plan to consider layer assignment targeting at delay and slew optimization concurrently while reducing buffering overhead. As in emerging technology nodes, the routing algorithm should be able to adapt the heterogeneous layer structures, we believe this paper will stimulate more research for timing improvement in advanced routing, and shed more light on traditional EDA topics.

ACKNOWLEDGMENT

The authors would like to thank S. Hu at Michigan Technological University and W.-H. Liu at Cadence Design Systems for helpful discussions and comments.

REFERENCES

[1] J. H.-C. Chen, T. E. Standaert, E. Alptekin, T. A. Spooner, and V. Paruchuri, "Interconnect performance and scaling strategy at 7 nm node," in *Proc. IEEE Int. Interconnect Technol. Conf. (IITC)*, San Jose, CA, USA, 2014, pp. 93–96.

[2] J. Cong, "An interconnect-centric design flow for nanometer technologies," *Proc. IEEE*, vol. 89, no. 4, pp. 505–528, Apr. 2001.

[3] M. Cho and D. Z. Pan, "BoxRouter: A new global router based on box expansion and progressive ILP," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 26, no. 12, pp. 2130–2143, Dec. 2007.

[4] J. A. Roy and I. L. Markov, "High-performance routing at the nanometer scale," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 27, no. 6, pp. 1066–1077, Jun. 2008.

[5] T.-H. Wu, A. Davoodi, and J. T. Linderoth, "GRIP: Global routing via integer programming," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 30, no. 1, pp. 72–84, Jan. 2011.

[6] M. D. Moffitt, "MaizeRouter: Engineering an effective global router," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 27, no. 11, pp. 2017–2026, Nov. 2008.

[7] C.-H. Hsu, H.-Y. Chen, and Y.-W. Chang, "Multilayer global routing with via and wire capacity considerations," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 29, no. 5, pp. 685–696, May 2010.

[8] Y.-J. Chang, Y.-T. Lee, J.-R. Gao, P.-C. Wu, and T.-C. Wang, "NTHU-route 2.0: A robust global router for modern designs," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 29, no. 12, pp. 1931–1944, Dec. 2010.

[9] W.-H. Liu, W.-C. Kao, Y.-L. Li, and K.-Y. Chao, "NCTU-GR 2.0: Multithreaded collision-aware global routing with bounded-length maze routing," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 32, no. 5, pp. 709–722, May 2013.

[10] (Nov. 2014). *ITRS*. [Online]. Available: http://www.itrs.net

[11] M.-K. Hsu *et al.*, "Design and manufacturing process co-optimization in nano-technology," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, San Jose, CA, USA, 2014, pp. 574–581.

[12] Z. Li *et al.*, "Fast interconnect synthesis with layer assignment," in *Proc. ACM Int. Symp. Phys. Design (ISPD)*, Portland, OR, USA, 2008, pp. 71–77.

[13] S. Hu *et al.*, "Fast algorithms for slew-constrained minimum cost buffering," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 26, no. 11, pp. 2009–2022, Nov. 2007.

[14] S. Hu, Z. Li, and C. J. Alpert, "A polynomial time approximation scheme for timing constrained minimum cost layer assignment," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, San Jose, CA, USA, 2008, pp. 112–115.

[15] S. Hu, Z. Li, and C. J. Alpert, "A faster approximation scheme for timing driven minimum cost layer assignment," in *Proc. ACM Int. Symp. Phys. Design (ISPD)*, San Diego, CA, USA, 2009, pp. 167–174.

[16] T.-H. Lee and T.-C. Wang, "Congestion-constrained layer assignment for via minimization in global routing," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 27, no. 9, pp. 1643–1656, Sep. 2008.

[17] K.-R. Dai, W.-H. Liu, and Y.-L. Li, "NCTU-GR: Efficient simulated evolution-based rerouting and congestion-relaxed layer assignment on 3-D global routing," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 20, no. 3, pp. 459–472, Mar. 2012.

[18] W.-H. Liu and Y.-L. Li, "Negotiation-based layer assignment for via count and via overflow minimization," in *Proc. IEEE/ACM Asia South Pac. Design Autom. Conf. (ASPDAC)*, Yokohama, Japan, 2011, pp. 539–544.

[19] J. Ao, S. Dong, S. Chen, and S. Goto, "Delay-driven layer assignment in global routing under multi-tier interconnect structure," in *Proc. ACM Int. Symp. Phys. Design (ISPD)*, Stateline, NV, USA, 2013, pp. 101–107.

[20] T.-H. Lee and T.-C. Wang, "Simultaneous antenna avoidance and via optimization in layer assignment of multi-layer global routing," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, San Jose, CA, USA, 2010, pp. 312–318.

[21] T.-H. Lee, Y.-J. Chang, and T.-C. Wang, "An enhanced global router with consideration of general layer directives," in *Proc. ACM Int. Symp. Phys. Design (ISPD)*, Santa Barbara, CA, USA, 2011, pp. 53–60.

[22] D. Liu, B. Yu, S. Chowdhury, and D. Z. Pan, "Incremental layer assignment for critical path timing," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, Austin, TX, USA, 2016, pp. 1–6.

[23] D. Liu, B. Yu, S. Chowdhury, and D. Z. Pan, "Incremental layer assignment for timing optimization," *ACM Trans. Design Autom. Electron. Syst.*, vol. 22, no. 4, p. 75, 2017.

[24] V. Livramento *et al.*, "Incremental layer assignment driven by an external signoff timing engine," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 36, no. 7, pp. 1126–1139, Jul. 2017.

[25] Y. Peng and X. Liu, "Low-power repeater insertion with both delay and slew rate constraints," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, San Francisco, CA, USA, 2006, pp. 302–307.

[26] Y. Zhang, A. Chakraborty, S. Chowdhury, and D. Z. Pan, "Reclaiming over-the-IP-block routing resources with buffering-aware rectilinear Steiner minimum tree construction," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, San Jose, CA, USA, 2012, pp. 137–143.

[27] O. Coudert, J. Cong, S. Malik, and M. Sarrafzadeh, "Incremental CAD," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, San Jose, CA, USA, 2000, pp. 236–244.

[28] C. Albrecht, "Efficient incremental clock latency scheduling for large circuits," in *Proc. IEEE/ACM Design Autom. Test Eurpoe (DATE)*, Munich, Germany, 2006, pp. 1–6.

[29] H.-Y. Chang, I. H.-R. Jiang, and Y.-W. Chang, "Timing ECO optimization via Bézier curve smoothing and fixability identification," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 31, no. 12, pp. 1857–1866, Dec. 2012.

[30] S. K. Karandikar *et al.*, "Fast electrical correction using resizing and buffering," in *Proc. IEEE/ACM Asia South Pac. Design Autom. Conf. (ASPDAC)*, Yokohama, Japan, 2007, pp. 553–558.

[31] S. Roy, P. M. Mattheakis, L. Masse-Navette, and D. Z. Pan, "Clock tree resynthesis for multi-corner multi-mode timing closure," in *Proc. ACM Int. Symp. Phys. Design (ISPD)*, Petaluma, CA, USA, 2014, pp. 69–76.

[32] J. A. Roy and I. L. Markov, "ECO-system: Embracing the change in placement," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 26, no. 12, pp. 2173–2185, Dec. 2007.

[33] T. Luo *et al.*, "Pyramids: An efficient computational geometry-based approach for timing-driven placement," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, San Jose, CA, USA, 2008, pp. 204–211.

[34] Y. Zhang and C. Chu, "GDRouter: Interleaved global routing and detailed routing for ultimate routability," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, San Francisco, CA, USA, 2012, pp. 597–602.

[35] C.-H. Hsu, H.-Y. Chen, and Y.-W. Chang, "Multi-layer global routing considering via and wire capacities," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, San Jose, CA, USA, 2008, pp. 350–355.

[36] C. V. Kashyap, C. J. Alpert, F. Liu, and A. Devgan, "Closed form expressions for extending step delay and slew metrics to ramp inputs," in *Proc. ACM Int. Symp. Phys. Design (ISPD)*, Monterey, CA, USA, 2003, pp. 24–31.

[37] N. J. Naclerio, S. Masuda, and K. Nakajima, "The via minimization problem is NP-complete," *IEEE Trans. Comput.*, vol. 38, no. 11, pp. 1604–1608, Nov. 1989.

[38] A. P. Ruszczyński, *Nonlinear Optimization*, vol. 13. Princeton, NJ, USA: Princeton Univ. Press, 2006,

[39] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms, and Applications*. Beijing, China: Pearson, 2005.

[40] R. G. Michael and S. J. David, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman, 1979.

[41] M. Queyranne, "Performance ratio of polynomial heuristics for triangle inequality quadratic assignment problems," *Oper. Res. Lett.*, vol. 4, no. 5, pp. 231–234, 1986.

[42] A. D. Gunawardena, S. Jain, and L. Snyder, "Modified iterative methods for consistent linear systems," *Linear Algebra Appl.*, vols. 154–156, pp. 123–143, Aug./Oct. 1991.

[43] *LEMON: Library for Efficient Modeling and Optimization in Networks*. Accessed: Nov. 2016. [Online]. Available: http://lemon.cs.elte.hu/trac/lemon

[44] (2015). *OpenMP: An Industry-Standard API for Shared-Memory Programming*. [Online]. Available: http://www.openmp.org/

[45] G.-J. Nam, C. Sze, and M. Yildiz, "The ISPD global routing benchmark suite," in *Proc. ACM Int. Symp. Phys. Design (ISPD)*, Portland, OR, USA, 2008, pp. 156–159.

Derong Liu received the B.S. degree in microelectronics from Fudan University, Shanghai, China, in 2011. She is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, University of Texas at Austin, Austin, TX, USA, under the supervision of Prof. D. Z. Pan.

Her current research interests include physical design and design automation for logic synthesis.

Bei Yu (S'11–M'14) received the Ph.D. degree from the Department of Electrical and Computer Engineering, University of Texas at Austin, Austin, TX, USA, in 2014.

He is currently an Assistant Professor with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong. His current research interests include combinatorial algorithm and machine learning with applications in very large scale integration computer aided design, and cyber-physical systems.

Dr. Yu has served in the Editorial Boards of Integration, *Very Large Scale Integration (VLSI) Journal* and *IET Cyber-Physical Systems: Theory & Applications*.

Salim Chowdhury received the Ph.D. degree from the University of Southern California, Los Angeles, CA, USA, in 1986.

He taught at the University of Iowa, Iowa City, IA, USA, for some years, where he obtained multiple research grants from National Science Foundation. He has been with semiconductor industries since then, with Motorola, Chicago, IL, USA, Sun Microsystem, Santa Clara, CA, USA, and until recently with Oracle America, Redwood City, CA, USA.

Dr. Chowdhury was recipient of the Best Paper Award from DAC and holds many patents and publications.

David Z. Pan (S'97–M'00–SM'06–F'14) received the Ph.D. degree from the University of California at Los Angeles (UCLA), Los Angeles, CA, USA, in 2000.

From 2000 to 2003, he was a Research Staff Member with IBM T. J. Watson Research Center, Yorktown Heights, NY, USA. He is an Engineering Foundation Professor with the Department of Electrical and Computer Engineering, University of Texas at Austin, Austin, TX, USA. His current research interests include cross-layer design for manufacturability, reliability, security, physical design, and CAD for emerging technologies. He has published over 280 papers and is the holder of eight U.S. patents.

Dr. Pan was a recipient of the SRC 2013 Technical Excellence Award, the DAC Top 10 Author in Fifth Decade, the DAC Prolific Author Award, the ASPDAC Frequently Cited Author Award, 14 Best Paper Awards, Communications of the ACM Research Highlights in 2014, the ACM/SIGDA Outstanding New Faculty Award in 2005, the NSF CAREER Award in 2007, the SRC Inventor Recognition Award three times, the IBM Faculty Award four times, the UCLA Engineering Distinguished Young Alumnus Award in 2009, the UT Austin RAISE Faculty Excellence Award in 2014, and many international CAD contest awards.