Incremental Layer Assignment for Timing Optimization

DERONG LIU, The University of Texas at Austin, Austin, Texas
BEI YU, The Chinese University of Hong Kong, Hong Kong
SALIM CHOWDHURY, Austin, Texas
DAVID Z. PAN, The University of Texas at Austin, Austin, Texas

With VLSI technology nodes scaling into the nanometer regime, interconnect delay plays an increasingly critical role in timing. For layer assignment, most works deal with via counts or total net delays, ignoring critical paths of each net and resulting in potential timing issues. In this article, we propose an incremental layer assignment framework targeting delay optimization in timing the critical path of each net. A set of novel techniques are presented: self-adaptive quadruple partition based on K×K division benefits the runtime; semidefinite programming is utilized for each partition; and the sequential mapping algorithm guarantees integer solutions while satisfying edge capacities; additionally, concurrent mapping offers a global view of assignment and post delay optimization reduces the path timing violations. The effectiveness of our work is verified by ISPD'08 benchmarks.

Categories and Subject Descriptors: B.7.2 [**Hardware, Integrated Circuit**]: Design Aids

General Terms: Design, Algorithms, Performance

Additional Key Words and Phrases: Layer assignment, critical path timing, semidefinite programming

ACM Reference Format:
Derong Liu, Bei Yu, Salim Chowdhury, and David Z. Pan. 2017. Incremental layer assignment for timing optimization. ACM Trans. Des. Autom. Electron. Syst. 22, 4, Article 75 (June 2017), 25 pages.
DOI: http://dx.doi.org/10.1145/3083727

1. INTRODUCTION

In emerging technology nodes, transistor and interconnect feature sizes are scaling further into the nanometer regime, and thus timing issues on interconnects are dominant in modern design closure [Chen et al. 2014]. As an integral part of the timing convergence flow, global routing determines the topologies of all nets and thus is critical for performance optimization [Hu and Sapatnekar 2001].

As a key step of global routing, layer assignment is important for assigning net segments into appropriate metal layers. Many metrics should be considered during layer assignment, such as via counts, congestion, timing issues, and so forth. Since each net may have one or several timing paths, layer assignment should also pay attention to the segments on these critical paths to avoid potential timing violations. Besides, in advanced technology nodes, resistance and capacitance values vary significantly

This work is supported in part by NSF SRC, NSFC, Oracle, and the Chinese University of Hong Kong (CUHK) Direct Grant for Research.
Authors' addresses: D. Liu and D. Z. Pan, Department of Electrical and Computer Engineering, The University of Texas at Austin, TX USA; emails: {deronliu, dpan}@cerc.utexas.edu; B. Yu, Department of Computer Science and Engineering, The Chinese University of Hong Kong, NT, Hong Kong; email: byu@cse.cuhk.edu.hk; S. Chowdhury, Austin, TX USA; email: salimuc@gmail.com.

DOI: http://dx.doi.org/10.1145/3083727

among different metal layers [Hsu et al. 2014]: higher metal layers are wider with lower resistance, while lower metal layers are thinner with higher resistance values. Thus, high layers are more attractive for timing critical nets that may introduce serious timing issues. Nevertheless, since there exist edge capacity constraints for edges on global routing grids for each metal layer, not all segments are allowed to be assigned on higher layers. The segments leading to critical sinks of a net are preferred to be assigned on high metal layers to reduce the potential timing violations. Therefore, an intelligent layer assignment framework is necessary to reduce the critical path timing.

There are many layer assignment works, targeting minimization of via count, antenna effect avoidance, timing optimization, and so forth [Lee and Wang 2008; Dai et al. 2009; Liu and Li 2011; Lee and Wang 2010; Ao et al. 2013; Yu et al. 2015a; Shi et al. 2016]. For via count minimization, a polynomial-time algorithm determines the net order and then solves one net each time through dynamic programming considering congestion issues [Lee and Wang 2008]. Dai et al. [2009] also apply a dynamic programming for net-by-net layer assignment. However, the sequential net ordering lacks a global view and thus affects the final performance because nets with higher priorities have more layer selections while those nets with lower priorities lack better resources. To alleviate the net order limitation, Liu and Li [2011] adopt a negotiation-based methodology to minimize via count and capacity violations. Meanwhile, antenna avoidance is included during layer assignment where via counts are also reduced through the min-cost max-flow model [Lee and Wang 2010]. Ao et al. [2013] focus on optimizing via counts and net delay. Nevertheless, the via capacity model is not considered, and thus, more wires may be assigned on high metal layers, resulting in capacity violations. Very recently, Yu et al. [2015a] proposed an incremental layer assignment integrated with a timing optimization engine. The proposed framework, TILA, is able to provide a global view of minimizing the total net delay for the selected nets. As an extension, Liu et al. [2017] additionally reduce the slew violations with a control of via overheads. Also, Livramento et al. [2017] guide the global optimization of timing critical paths by decoupling the layer assignment from timing analysis.

Although TILA [Yu et al. 2015a] can achieve the state-of-the-art layer assignment results targeting timing optimization, it may still suffer from the following shortcomings: (1) The optimization engine of TILA is based on Lagrangian relaxation, whose performance may heavily rely on the initial values of multipliers. (2) In addition, when via delay and via capacity are considered, layer assignment is similar to a quadratic assignment problem [Queyranne 1986], which is essentially a nonlinear optimization problem. However, to achieve extremely fast speed, TILA artificially approximates some quadratic terms to a linear model, which may impact the layer assignment accuracy and performance. (3) Compared to TILA, we focus more on critical path timing in each net instead of the total sum of net delays.

In this article, we propose a novel incremental layer assignment framework targeting timing optimization for critical timing paths in nets, where our layer assignment tool is able to achieve better timing optimization. Figure 1 compares the layer assignment results between TILA and our work. Figure 1(a) gives the results from TILA, where many pins have a delay of over 4.2×10^6. On the other hand, from Figure 1(b), we can see that our framework can reduce the maximum delay since the worst pin has a delay around 4.2×10^6. The contributions of our work are listed as follows:

—An integer linear programming (ILP) formulation is presented to optimize the critical path delay of selected critical nets.
—A self-adaptive partitioning methodology based on $K \times K$ division benefits the runtime.
—A semidefinite programming (SDP) relaxation is adopted for further speedup with a post mapping methodology to guarantee integer solutions.

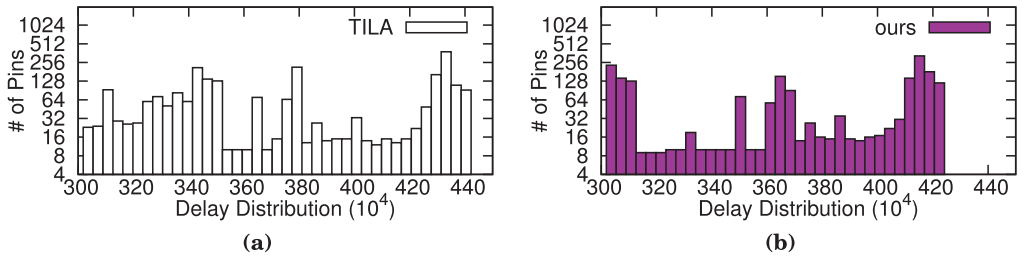

Fig. 1. Pin delay distribution of critical nets for benchmark adaptec1, where 0.5% of the nets are released as critical nets. (a) Results from TILA [Yu et al. 2015a]. (b) Results from our incremental layer assignment framework.

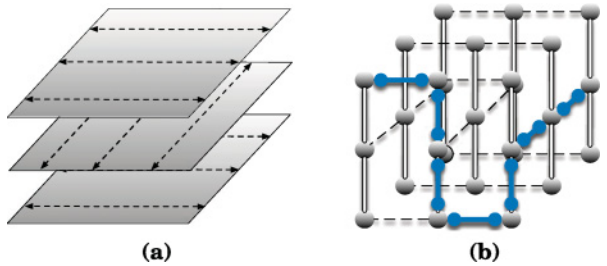

Fig. 2. Layer and grid model. (a) Layer model. (b) Net routing on grid model.

—A concurrent matching flow is attached to provide more concrete solutions for SDP results, followed by a post delay optimization algorithm.

The remainder of this article is organized as follows. In Section 2, we provide some preliminaries and the problem formulation. In Section 3, we first present the mathematical formulation to optimize critical path timing. Then we discuss a set of novel techniques to further achieve a better tradeoff between solution quality and runtime. In Section 4, we report the experimental results, followed by the conclusion in Section 5.

2. PRELIMINARIES

2.1. Graph Model

Figure 2(a) shows a layer model, where each layer supports unidirectional wires, either in the horizontal or vertical direction, and the dotted lines represent the preferred routing directions for each layer. Based on this, the layer assignment problem can be modeled on a 3-dimensional grid graph [Hu and Sapatnekar 2001], as shown in Figure 2(b). We can see that a layer is divided into a large set of rectangular tiles, represented by the vertices in the grid model. Furthermore, the edges connecting vertices are divided into two sets: edges in the x/y-direction are for routing wires on layers and edges in the z-direction are for vias between layers. Figure 2(b) shows a net routing on the three-layer grid, which consists of segments and vias along the edges.

For x/y-direction edges, each of them has a specified routing capacity on different layers, that is, $cap_e(l)$ for each layer l. This is to say that the number of wires placed on layer l of this edge should not be higher than $cap_e(l)$. Figure 3(a) provides a detailed illustration of edge capacity model, where the number of wires passing on Metal 2, that is, M2, should not exceed four. Notably, for an incremental layer assignment tool, considering that the nonreleased segments also occupy the routing resources, we should deduct these segments from the original $cap_e(l)$ so that the total number of passing wires cannot exceed the number of physical tracks. Therefore, as seen in Figure 3(a), when there is a non-released segment routed on the edge marked as blue

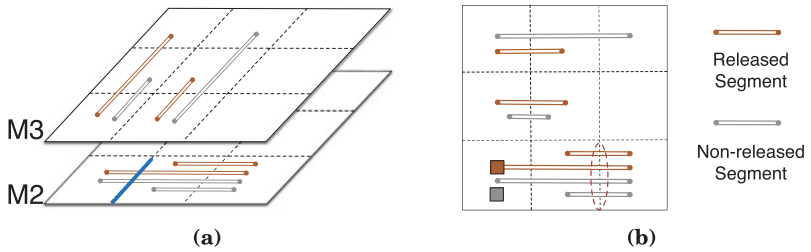

Fig. 3. Illustration of capacity model. (a) Edge capacity model. (b) Via capacity model.

on M2, the current edge capacity, $cap_e(l)$, should be set to three for released segments to assign.

Similarly, there is also a specified via capacity constraint for vias passing through each routing grid. The via capacity constraint is determined by the available routing capacity of two edges associated with this vertex. Additionally, since stacked vias are known to consume extra routing resources in each grid, here we mainly consider the overflow caused by stacked vias in our framework, the same as Hsu et al. [2008], and the impact of nonstacked vias will be taken into account as a future work. To make it explicit, an illustration of via capacity model is shown in Figure 3(b). Meanwhile, we should also take care of those nonreleased vias based on the original via capacity. In fact, $cap_g(l)$ represents the available routing via spaces for those released nets in our framework. Therefore, for the lower left grid in Figure 3(b), the gray square represents a via from a nonreleased net, so its occupied area should be counted for residual via capacity constraints. As layer assignment works as an important step in global routing, the available via capacity is computed as follows [Hsu et al. 2008]:

$$cap_g(l) = \left\lfloor \frac{(w_w + w_s) \cdot Tile_w \cdot (rcap_{e_0}(l) + rcap_{e_1}(l))}{2 \cdot (v_w + v_s)^2} \right\rfloor - n'_v, \tag{1}$$

where w_w, w_s, v_w, v_s, $Tile_w$ represent wire width, wire spacing, via width, via spacing, and tile width, respectively. n'_v denotes the number of vias from nonreleased nets. For vias between two layers, each layer has two edges connecting with grid g, that is, e_0 and e_1. Their available routing capacities are represented by $rcap_{e_0}(l)$, $rcap_{e_1}(l)$, respectively, which are the residual available routing tracks considering the assignments of segments. Different from $cap_e(l)$ earlier, which does not consider released segments on edge e, here $rcap_e(l)$ provides the exact residual edge capacity including both released and nonreleased segments. Therefore, from Equation (1), we can see that the allowable via number in each tile should not exceed the residual space divided by the via area. This means the resulting stacked vias can only take advantage of the residual routing spaces after all the wires have been routed. In Figure 3(b), for the circled edge, its $cap_e(2)$ is set to 2 but $rcap_e(2)$ is set to 0 since two released segments occupy the left routing resources. Then no vias are allowed to pass through this grid because two connected edges are full to the capacity, that is, no residual space.

2.2. Timing Model

To calculate the timing cost of each net, we adopt the *Elmore* delay model, which is generally utilized to estimate the wire delay during timing analysis. The timing costs consist of segment delays and via delays, both of which depend on the layer resistance and their corresponding downstream capacitance. Equation (2) gives the timing cost calculation of segment s_i on layer l:

$$t_s(i, l) = R_e(l) \cdot (C_e(l)/2 + C_d(i)), \tag{2}$$

where $R_e(l), C_e(l)$ refer to the wire resistance and capacitance of the layer l on which segment i passing edge e is assigned, and $C_d(i)$ is the downstream capacitance of segment i. It is seen that the timing cost of s_i depends on its downstream capacitance $C_d(i)$ and the corresponding resistance/capacitance values of its assigned layer. During calculating $C_d(i)$, the layer assignments of all downstream segments of segment s_i should be taken into account. Thus, we compute $C_d(i)$ from sinks to source in a bottom-up manner based on the *Elmore* delay model. Besides, $R_e(l), C_e(l)$ can be computed from specified technology libraries. With these provided parameters, the timing cost of segment s_i for the *Elmore* delay is obtained.

Similarly, via timing cost is calculated as in Equation (3), which is determined by via resistance and the downstream capacitance of its connected segments [Yu et al. 2015a]:

$$t_v(i, j, p, q) = \sum_{l=j}^{q-1} R_v(l) \cdot C_d(V(s_i, s_p)), \tag{3}$$

where segment s_i on layer j is connected with segment s_p on layer q, $R_v(l)$ is the resistance of via between layers l and $l + 1$, and we assume layer j is lower than layer q, while $V(s_i, s_p)$ corresponds to the set of stacked vias connecting segment s_i and segment s_p. Thus, the calculation of via delay mainly depends on via resistances between layers and its corresponding downstream capacitance. In this work, the via downstream capacitance, that is, $C_d(V(s_i, s_p))$, is equal to that of its upstream segment, which refers to the segment closer to the net driver, because via capacitance is not considered in this work. Additionally, for vias through multiple layers, it is required to add the via delay between two adjacent layers from the lowest to the highest layer. Therefore, the integration of via delay is also able to benefit via consumptions.

2.3. Problem Formulation

Based on the grid model and timing model discussed in the preceding section, we define the critical path layer assignment (CPLA) problem as follows:

Problem 1 (*CPLA*). Given a 3D grid graph, edge and layer information, initial routing and layer assignment, and a set of critical nets, layer assignment reassigns layers among critical and noncritical nets sharing metal resources onto layers in order to minimize their critical path timing while satisfying the edge capacity constraints.

3. CPLA ALGORITHMS

In this section, we discuss the details of our framework to solve the CPLA problem. First, we propose an integer linear programming (ILP) formulation. Then we relax this formulation into semidefinite programming (SDP). To make this problem solvable for SDP, a self-adaptive quadruple partitioning methodology is also presented to select appropriate problem sizes for SDP. Then, we give the sequential mapping algorithm to locate integer solutions, and a further concurrent matching strategy follows for better optimization. Finally, we present a post delay optimization step to reduce potential path timing violations.

3.1. ILP Formulation

As an incremental layer assignment work, similar to TILA, we take an initial solution as an input and release a certain ratio of nets to be optimized. This ratio is termed "critical ratio," which determines the problem size intuitively. From the perspective of timing optimization, we prefer to locate those critical nets on high and thick layers, while the same ratio of nets with good timing will also be released and reassigned on

low layers to provide the required routing resources. These nets are termed "noncritical nets." To make it explicit, both critical and noncritical nets will be reassigned in our framework. Based on the grid model shown in Figure 2(b), each net is composed of a sequence of segments that have the same length as a routing grid. Thus, those segments belonging to "critical nets" are denoted as "critical segments," and vice versa.

It is seen that CPLA utilizes a similar framework as TILA but distinguishes from TILA in the following aspects: First, TILA cares about the sum of segments' delay in a net, while CPLA focuses on each net's worst timing path. As a single net can be divided into a set of paths, where each path corresponds to a single sink, the sink with the worst timing overhead is identified as the critical sink, and its connected path is this net's critical path. Then the maximum path timing of each net can be acquired through the delay of its critical sink. Here we do not take the cell information into account, so we focus on the maximum path timing of each critical net. Therefore, during the selection of critical nets, we choose those with the worst maximum path timing rather than total delays. Second, the same ratio of noncritical nets should also be selected to release high layer resources for those critical nets. Instead of optimizing all the critical and noncritical nets simultaneously in TILA, CPLA places more emphasis on those critical nets that will be assigned at first. After the assignment of critical nets, a greedy method is adopted to assign these noncritical nets in a one-by-one way. Since our optimal target is the maximum path timing of those critical nets, the assignment details of noncritical nets will not be covered, but we follow a dynamic programming method in order to control the via counts in Lee and Wang [2008]. Through optimizing critical and noncritical nets separately, CPLA is able to provide sufficient high layer resources for those critical nets to achieve better timing.

During the selection of critical nets, we measure the maximum path timing of each net and select those with the worst values based on the specified ratio. Then, to select a set of noncritical nets efficiently, we should search for those nets with the best timing that also share the same edges with the critical ones as much as possible. And the assigned layers of the noncritical nets should be higher than critical nets for resource releasing. In summary, the selection procedure of noncritical nets is similar to Yu et al. [2015a], but with one main difference: here we select the critical/noncritical nets based on their maximum path timing instead of their total sum delays. Therefore, with the maximum path timing and given critical ratio, a set of critical nets and noncritical nets are selected for reassignment. More details of our proposed ILP formulation are given as follows, while noncritical nets are not included in this formulation. As introduced in Section 2, segment delay can be calculated based on Equation (2), and via delay based on Equation (3). For convenience, notations used are listed in Table I.

Thus, we can obtain the ILP formulation as shown in Equation (4). This formulation concerns all the segments and vias along the critical timing paths in all critical nets, and also contains the branches due to the fact that they would affect the downstream capacitance of the maximum path.

In our mathematical formulation, the constraint in Equation (4b) guarantees that one segment can be assigned on one and only one layer. The constraint in Equation (4c) sets the routing wire limit for those released segments of edge e on layer j, that is, $cap_e(j)$. Notably, $cap_e(l)$ not only depends on its initial track number but also those nonreleased segments passing through e on layer l. As shown in Figure 3(a), when the blue edge on M2 has four available tracks initially but one of them has already been occupied by a nonreleased net, the exact allowable routing capacity, that is, $cap_e(2)$, should be set to 3. This means that at most three wires are allowed to be routed through this edge for the released nets. In this way, we can see that $cap_e(l)$ may vary for each edge e on the same layer, due to the variance of existing nonreleased nets. Thus, for an incremental assignment problem, the edge capacity constraint is more stringent than the initial

Table I. Notations Used for ILP Formulation

Nc	set of all critical nets
L	set of all layers
S	set of all segments
E	set of all edges in the whole grid model
G	set of global routing grids
$S(Nc)$	set of all segments for all critical nets Nc
$Sx(Nc)$	set of all pairs of segments of critical nets Nc while two segments in a pair are being connected by one or more vias
$Sx(Nc, g)$	set of all pairs of segments of critical nets Nc passing through one routing grid g
S_e	set of released critical segments on edge $e \in E$
$V(s_i, s_p)$	set of vias connecting critical segments s_i and segment s_p
x_{ij}	binary variable, set to 1 if segment s_i is assigned to layer j
$t_s(i, j)$	timing cost when critical segment s_i is assigned to layer j
y_{ijpq}	binary variable, set to 1 if both x_{ij} and x_{pq} are set to 1
$t_v(i, j, p, q)$	timing cost for vias in $V(s_i, s_p)$ from layer j to q
$cap_e(l)$	available routing capacity of edge e on layer l for released segments
$rcap_e(l)$	residual routing capacity of edge e on layer l after routing all nets
$cap_g(l)$	available via capacity of node g on layer l for released nets

problem.

$$\textbf{min} \sum_{i \in S(Nc)} \sum_{j=1}^{L} t_s(i, j) \cdot x_{ij} + \sum_{i,p \in Sx(Nc)} \sum_{j=1}^{L-1} \sum_{q=1}^{L-1} t_v(i, j, p, q) \cdot y_{ijpq}, \tag{4a}$$

$$\textbf{s.t.} \sum_{j} x_{ij} = 1, \qquad \forall i \in S(Nc), \tag{4b}$$

$$\sum_{i \in S(e)} x_{ij} \leq cap_e(j), \qquad \forall e \in E, \tag{4c}$$

$$\sum_{(i,p) \in Sx(Nc,g)} y_{ijpq} + n_v(x_{ij} + x_{pq}) \leq cap_g(l), \qquad \forall l, j < l < q, \quad g \in G, \tag{4d}$$

$$x_{ij} \geq y_{ijpq}, \qquad \forall (i, p) \in Sx(Nc), \quad j, q \in L, \tag{4e}$$

$$x_{pq} \geq y_{ijpq}, \qquad \forall (i, p) \in Sx(Nc), \quad j, q \in L, \tag{4f}$$

$$x_{ij} + x_{pq} \leq y_{ijpq} + 1, \qquad \forall (i, p) \in Sx(Nc), \quad j, q \in L, \tag{4g}$$

$$y_{ijpq} \text{ is binary}, \qquad \forall (i, p) \in Sx(Nc), \quad j, q \in L, \tag{4h}$$

$$x_{ij} \text{ is binary}, \qquad \forall i \in S(Nc), \quad j \in L. \tag{4i}$$

Considering the possible existing edge overflows from the input, we extend this constraint to comply for both legal and illegal solutions. For legal solutions free of overflows, it is clear to keep $cap_e(l)$ as noted earlier, that is, the initial number of edge capacity excluding those nonreleased segments; however, for those solutions with edge overflows, we increase $cap_e(l)$ to accommodate the routing wires accordingly. This is to say, for an edge e on layer l, if the input solution provides five routing wires but its capacity should be 4, then an edge overflow does exist. To deal with that, if there are two nonreleased segments on it, then $cap_e(l)$ will be set to 3 for those released segments and no further edge overflows will be produced.

Similarly, the constraint in Equation (4d) places the limitation of the via number to pass through each grid g for different layers. Similar to $cap_e(l)$, the calculation of $cap_g(l)$ should take those nonreleased nets into consideration as well. Still, as shown by the lower left grid in Figure 3(b), we assume that the initial via capacity for this grid from M2 to M3 is 16, where each track is able to locate four vias and there are in total four tracks. Meanwhile, there is already a nonreleased via passing through M2 and one track is also occupied by another nonreleased net. Then we reduce the available via space further by deducting the utilized resources of nonreleased nets, that is, five vias in total. Notably, a newly assigned segment will also take another available track for routing, thus resulting in further reduction of four vias. The final residual space is for routing vias belonging to those released segments. Therefore, in the constraint in Equation (4d), we should consider not only those existing nonreleased nets but also the newly assigned segments. Take Figure 3(b) as an example; at most seven stacked vias are allowed to be inserted from released segments for the lower left grid. Through this setting, our framework is able to provide an estimation of the number of allowable vias for an incremental approach.

In Equation (4a), y_{ijpq} represents the via connecting segment s_i on layer j and segment s_p on layer q. Affected by x_{ij} and x_{pq}, y_{ijpq} should be set to 1 if and only if both x_{ij} and x_{pq} are set to 1 simultaneously. Therefore, y_{ijpq} can be understood as the product of x_{ij} and x_{pq}. Then in the constraints in Equations (4e) to (4g) y_{ijpq} is the product of x_{ij} and x_{pq} because all x_{ij} and y_{ijpq} are binaries according to the constraints in Equations (4h) and (4i).

Nevertheless, there is a potential problem for the constraint in Equation (4d). If via capacity violations already exist in the input solution and cannot be eliminated completely, this constraint may be too stringent that no legal solutions can be obtained. To avoid this case, we relax this constraint by adding a slack variable V_o, representing the number of maximum allowable violations. Then the constraint in Equation (4d) can be rewritten as follows:

$$\sum_{(i,p)\in Sx(Nc,g)} y_{ijpq} + n_v \cdot (x_{ij} + x_{pq}) \leq cap_g(l) + V_o, \forall l, j < l < q, \forall g \in G.$$

V_o is considered in the objective formulation with a weighting parameter α, which is set to 2, 000 in our implementation. Thus, the ILP formulation can guarantee reasonable solutions with legal edge capacities and controllable via violations. Similar to Yu et al. [2015a], our framework solves layer assignment through an iterative scheme and stops when no further optimizations can be achieved. However, for large benchmarks, ILP could lead to a huge calculation overhead with considerable runtime. In order to alleviate this overhead, speedup techniques are introduced in the following sections.

3.2. Self-Adaptive Partition Algorithm

For layer assignment work, the routing wires are adjusted in the z-dimension among different layers. Thus, the whole grid model can be divided into $K \times K$ partitions in x/y-dimensions, and each division is solved separately from its neighbors. Also, as mentioned in Gunawardena et al. [1991], the newly updated assignment results of neighboring partitions benefit each current partition. Figure 4(a) gives examples of several nets to be divided by 3×3 divisions, which are identified with different colors. Through partitioning, the problem size can be reduced by $\frac{1}{K\times K}$ times on average. However, Figure 4(b) shows that the routing congestion density varies significantly for each division. Here various colors imply the routing distribution of nets passing through these regions. We can see that uniform division by $K \times K$ may lead to unbalanced computing resource allocation among these congested regions and those marginal regions

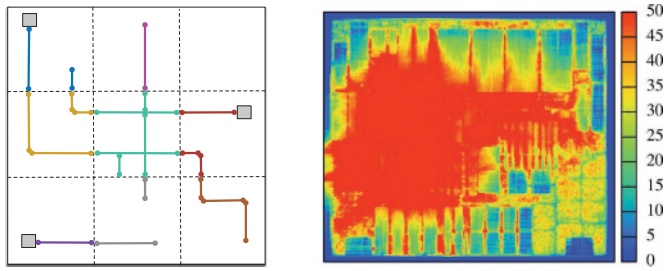

Fig. 4. Example of grid partition. (a) Nets partition. (b) Routing density for benchmark adaptec1 by NCTU-GR.

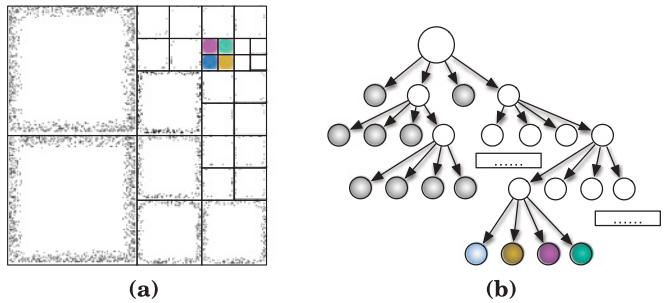

Fig. 5. Subgrid partition illustration. (a) Subgrid partition. (b) Subgrid corresponding partition tree.

containing fewer routing nets. Therefore, we propose a self-adaptive quadruple partition algorithm to further divide all $K \times K$ regions so that each region contains a similar number of critical segments.

Figure 5(a) gives the example of partition results for the lower left one in 5×5 divisions, where each division contains a similar number of critical segments. Here we limit the allowable maximum number of critical segments in each partition by setting a constraint. If the original division does not satisfy this constraint, then further partition operations are executed. Besides, Figure 5(b) shows the quadruple tree corresponding to Figure 5(a). If a partition has a small enough problem size, it will exist as a leaf node in the tree; otherwise, further quadruple partition continues until it meets the requirement. Note that for some dense regions, the constraint may be so tight that the number of segments on one edge may exceed the requirement, but no further partition should be allowed in fact. To avoid this, we also check if the current partition size is smaller than the tile width/height. If so, the partition should stop to avoid deadlocks.

After partitioning is completed, we obtain the leaf nodes as colored in Figure 5(a). There are two leaf nodes in the first level representing these two left partitions. In Figure 5(b), the bottom colored nodes represent four partitions with the same colors. With this partition methodology, we can adjust constraints to suit different algorithms efficiently. Furthermore, each partition can be solved in parallel with multiple threads. Since each of them has a similar problem size, each thread deals with a workload in a well-balanced manner.

3.3. Semidefinite Programming Relaxation

In the previous section, we propose a self-adaptive algorithm to partition the original problem to the appropriate size considering the density distribution. This provides

us an opportunity for further speedup. In our work, we relax this problem from ILP into semidefinite programming (SDP). SDP also contains a linear objective function constrained by linear equations, similar to Linear Programming (LP), but it is more general than LP due to its symmetric matrix forms. SDP is solvable in polynomial time and it provides a theoretically better solution than LP [Vandenberghe and Boyd 1996], and thus it has been applied in many circuit design problems, such as circuit sizing [Vandenberghe et al. 1997], high-level synthesis [Cong and Liu 2012], power/ground network optimization [Kahng et al. 2006; Du et al. 2011], and layout decomposition [Yu et al. 2015b; Kohira et al. 2015]. To the best of our knowledge, this is the first work to adopt SDP to solve the layer assignment problem. We rewrite the formulation into the following standard SDP form:

$$\textbf{min} \;\; \mathbf{T} \bullet \mathbf{X}, \tag{5a}$$

$$\textbf{s.t.} \;\; \mathbf{C} \bullet \mathbf{X} = \mathbf{b}, \tag{5b}$$

$$\mathbf{X} \succeq \mathbf{0}, \tag{5c}$$

where

$$\mathbf{T} \bullet \mathbf{X} = \sum_{i \in S(Nc)} \sum_{j \in L} T_{ij} X_{ij} = \text{trace}(\mathbf{T}^{\intercal} \mathbf{X}). \tag{6}$$

In Equation (5), matrices $\mathbf{T}$ and $\mathbf{X}$ are both $|S \cdot L|$-dimension symmetric matrices, where $|S|$ is the number of segments belonging to critical nets in each partition and $|L|$ is the number of layers. In Equation (6), $\mathbf{T} \bullet \mathbf{X}$ is the inner product of these two matrices $\mathbf{T}$ and $\mathbf{X}$. Besides, T_{ij} and X_{ij} are the entries lying in the ith row and the jth column of matrices $\mathbf{T}$ and $\mathbf{X}$, respectively.

Equation (7) shows all coefficients in matrix $\mathbf{T}$, where the items on the diagonal line represent the timing costs, that is, $t_s(i, j)$, for assigning segment i on layer j. Besides, $t_v(i, j, p, q)$ is the via cost on assigning segments i and p into layer j and layer q, respectively. Each $t_v(i, j, p, q)$ is in the same row as $t_s(i, j)$ and the same column as $t_s(p, q)$. Matrix $\mathbf{X}$ in Equation (8) gives the SDP solution to the layer assignment, where each x_{ij} is on the diagonal line. Similarly, y_{ijpq} is in the same row as x_{ij} and the same column as x_{pq}.

$$\mathbf{T} = \begin{pmatrix} t_s(i,j) & \dots & t_v(i,j,p,q) \\ \dots & \dots & \dots \\ t_v(i,j,p,q) & \dots & t_s(p,q) \end{pmatrix}, \tag{7}$$

$$\mathbf{X} = \begin{pmatrix} x_{ij} & \dots & y_{ijpq} \\ \dots & \dots & \dots \\ y_{ijpq} & \dots & x_{pq} \end{pmatrix}. \tag{8}$$

For each x_{ij}, it is expected to be binary and placed in the diagonal line of objective matrix $\mathbf{X}$. If x_{ij} is equal to 1, then x_{ij}^2 is also 1; if x_{ij} is equal to 0, then its square form is also 0. The item y_{ijpq} needs to satisfy the constraints in Equations (4e) to (4g), which also apply for continuous solutions. Because the constraints in Equations (4e) to (4g) are mainly inequalities, extra slack variables are added into the objective matrix, for SDP cannot support inequality constraints. With these constraints, SDP considers via costs as quadratic terms (same as in Equation (4a)).

To guarantee an effective solution, the constraints in ILP Equation (4) should also be included in SDP through Equation (5b). The constraints in Equations (4b) and (4c) can be formulated into the constraints of SDP easily since they are linear constraints. As all the constraints are constructed in a similar way, here we mainly provide the

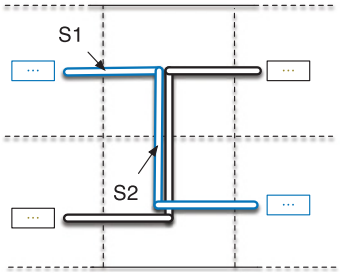

Fig. 6. An example of layer assignment through SDP.

$$\mathbf{T} = \begin{pmatrix} 35.2 & 0 & 5.8 & 6.7 \\ 0 & 15.6 & 2.3 & 3.5 \\ 5.8 & 2.3 & 47.8 & 0 \\ 6.7 & 3.5 & 0 & 23.9 \end{pmatrix} \qquad \mathbf{X} = \begin{pmatrix} 0.01 & 0 & 0 & 0 \\ 0 & 0.99 & 0.09 & 0.89 \\ 0 & 0.09 & 0.10 & 0 \\ 0 & 0.89 & 0 & 0.90 \end{pmatrix}$$

Fig. 7. $\mathbf{T}$ matrix and solution $\mathbf{X}$ matrix of the example.

details for the first constraint (Equation (4b)). Evidently, a set of coefficient matrices, $\mathbf{C_b}$s, is required, and the set size is equal to the number of segments in critical nets in each division. The dimension of each $\mathbf{C_b}$ is the same as $\mathbf{T}$ and $\mathbf{X}$. Thus, for each $\mathbf{C_b}$, according to the constraint in Equation (4b), we set each location in $\mathbf{C_b}$ corresponding to each x_{ij} in $\mathbf{X}$ as 1; meanwhile, the value of b in the right side of Equation (5b) is also 1. Through this setting, for segment s_i, the sum of x_{ij} is equal to 1 constrained by this equation. During construction of Equation (4c), because of the existences of inequalities, we require more slack variables in the objective matrix as the sum of variables should be smaller than the given edge capacity. The number of additional slack variables is equal to the number of edge capacity constraints. For the constraint in Equation (4d), we prefer to move it into the objective matrix by adding the penalty to save the runtime. Then penalty is represented as $\lambda_{i,j,p,q}$, which is added to $t_v(i, j, p, q)$ in matrix $\mathbf{T}$. The penalty is calculated by dividing the existing number of vias by its capacity.

To make it more clear, here we give an example of how SDP can be applied to the layer assignment problem. Figure 6 shows part of one net. Due to space limitations, we just focus on two segments, s_1 and s_2. We also assume there are only two available layers in each x/y-dimension: layer 1 and layer 3 for the x-dimension, and layer 2 and 4 for the y-dimension. Thus, the matrices $\mathbf{T}$ and $\mathbf{X}$ should both be 4×4 matrices, for each segment has two layers to assign. For convenience, we skip the slack matrices here because they are helping to satisfy the constraints. In our formulation, the entries on the diagonal line of matrix $\mathbf{T}$ are basically x_{ij}s, representing whether they are assigned on the corresponding layers. The entries in the same column and row with x_{ij} and x_{pq} represent the potential via costs from layer j to layer q. Based on Figure 6, s_1 only connects with s_2, so we just need to consider the via costs between s_1 and s_2.

In Figure 7, we list an example of matrix $\mathbf{T}$, as well as matrix $\mathbf{X}$ after solving the SDP. For matrix $\mathbf{X}$, four values on the diagonal line represent x_{11}, x_{13}, x_{22}, and x_{24}, respectively, where x_{ij} denotes segment s_i to be assigned on layer j. Thus, we see that s_1 should be assigned on layer 3 as x_{13} is very close to 1. Meanwhile, for s_2, both x_{22} and x_{24} are not so close to 1 because there is one segment released on the same edge. The edge capacity constraints may limit its value as floating points. In this case, we adopt a sequential mapping strategy to determine its layer to be assigned.

3.4. Sequential Mapping Algorithm

SDP provides us a continuous solution, which, however, cannot be applied to our problem directly. Therefore, an efficient mapping algorithm is necessary to provide discrete integer solutions while satisfying the stringent edge capacity constraints. In this section, we propose a sequential mapping algorithm to transfer a continuous SDP solution into a discrete layer assignment solution.

ALGORITHM 1: Sequential Mapping Algorithm

Input: Solution matrix $\mathbf{X}$;
1: Save entries (x_{ij}) for each segment i;
2: **for** each edge e containing critical segments **do**
3: **for** $j = L_m$; $j \geq 1$; $j = j - 2$ **do**
4: $n_{e_j} = cap_e(j)$;
5: Select n_{e_j} highest x_{ij}s on edge e;
6: Assign selected segment i on layer j;
7: Update $cap_e(j)$;
8: **end for**
9: **end for**

As stated, the basic idea of this sequential mapping algorithm is to map each continuous solution to an integer solution while satisfying the hard edge capacity constraints. Therefore, we should focus on each edge e on which some critical segments are assigned. Since a critical segment has a specified solution for each candidate layer of edge e, we should take advantage of the solution value to provide a reasonable assignment. By this way, we prefer to traverse each layer and select those segments endowed with the highest solutions on this layer, because these segments are most competitive for this layer. Due to the existing competition of high metal layers, we start from the highest layer for each edge and locate each segment based on their solutions.

The details of our mapping algorithm are shown in Algorithm 1, whose input is the original solution matrix $\mathbf{X}$. Initially, we read all the solution entries and save those x_{ij}s to each corresponding segment. Then we traverse each edge with these released segments in the whole grid (line 2) following the order from the highest layer to the lowest layer (line 3), for a higher metal layer has a lower resistance and is more competitive for segments to assign. Since edges are divided into x-dimensions and y-dimensions for different layers, we skip the layers containing all y-dimension edges for x-dimension edges and vice versa. As for layer j of edge e, there is a specified edge capacity constraint, that is, $cap_e(j)$. This means that the number of those released segments to assign should not exceed this constraint. Here we select the top $cap_e(j)$ entries and assign these segments to layer j (line 6). In this way, edge capacity overflows can be avoided based on the value of $cap_e(j)$. To avoid unnecessary conflicts, those segments that have been assigned on higher layers in previous iterations are skipped. In this way, the edge capacity constraint can be satisfied. Finally, the edge capacity is updated for this division. The runtime of this mapping algorithm is $\mathcal{O}(|E||L|log|S_e|)$, where $|E|$ is the number of edges with critical segments, $|L|$ is the number of layers, and $|S_e|$ is the number of critical segments on this edge.

3.5. Concurrent Matching Algorithm

The mapping algorithm proposed in Section 3.4 gives a sequential assignment of segments for different edges. The assignment is acquired based on the solutions from SDP, but for those segments whose solution values on one single layer are very close, their assignments may be a little coarse without considering detailed neighboring conditions. For instance, when both SDP solution values of two segments, that is, s_1 and s_2,

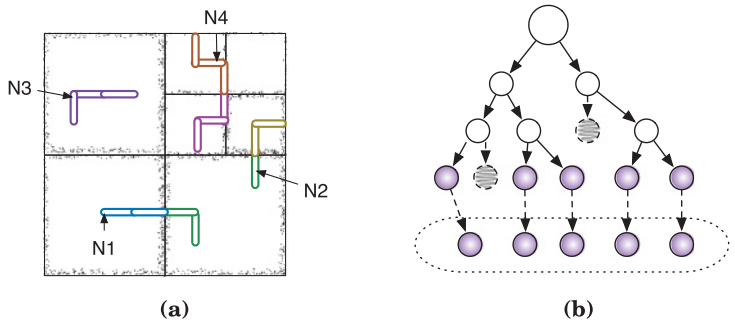

Fig. 8. Example of solution candidate generation. (a) Nets partition. (b) Solution candidate generation for N_3.

routed on the same edge, are equal to 0.5 for one layer but only one track is available, which segment to be assigned may be decided randomly. Although the impact of this assignment on timing may be slight, due to the closure of exact SDP results, the previous sequential algorithm lacks a global view of all nets to some extent. To handle this conflicting assignment, in this section we propose a concurrent matching methodology, which adopts new rounding strategies and targets more concrete solutions.

From the sequential algorithm, we obtain only one exact assignment for each edge based on noninteger solutions. Nevertheless, when the number of released segments on one edge is very high, keeping them on one assignment may lose some potential optimal solutions. In the following, we present how to produce more assignment candidates based on the SDP results.

As depicted in Section 3.2, the whole grid graph is partitioned into collections of divisions, and thus each division is able to possess one or more candidates. In one division, all the candidates are formed as a single solution set, while only one assignment is selected from this set as the result. Figure 8(a) provides an instance of four nets in different divisions. To make it explicit, all the segments in a division share the same color, even for multiple nets, like net N_1 and N_2. For net N_4, although some of its segments are on the boundary between two divisions, we assume these segments belong to their connecting left/bottom division. After making sure of the division details, we generate possible assignment candidates based on the SDP solutions.

Compared with the sequential mapping method, our concurrent matching flow mainly consists of two steps: candidate generation and solution mapping. The first step, how to generate a set of candidate solutions according to the floating-point solutions, deserves exploration. A procedure of candidate generation for segments in net N_3 is shown in Figure 8(b), where we adopt a top-down method to traverse all the possible assignments of net N_3. Since there are in total three segments for N_3, it is essential to seek three levels for promising solutions. For each reassigned segment, if its SDP value is smaller than a threshold, we will take its second-highest solution as its alternate assignment. The threshold is set to 0.9 here. To reduce the solution space, at most two possible layers are allowed for each segment to reassign. Even in this way, the whole solution size will get doubled after each segment has its second layer candidate. Furthermore, edge capacity checking is accompanied with each stage to guarantee the legality of possible solutions. As shown in Figure 8(b), when there is a violation, symbolized as a gray node, this intermediate solution will be abandoned. This stage guarantees that illegal solutions will be bounded beforehand and the solution candidates to be selected can satisfy the required capacity constraint. Through these two bounding methods, the solution space is controlled efficiently, and possibly

Table II. Notations for Post Stage Algorithms

D	set of all divisions containing segments
$D(Nc)$	set of all divisions containing critical nets Nc
$Dx(Nc)$	set of all pairs of divisions containing critical nets Nc
D_m	the m^{th} division in $D(Nc)$
$Sol(m)$	set of solutions of division $D_m \in D(Nc)$
a_{mn}	binary variable, set to 1 if the n^{th} solution is selected for D_m
b_{mnuv}	binary variable, set to 1 if the n^{th} solution is selected for D_m, and the v^{th} solution is selected for D_u
$c_d(m, n)$	cost when the n^{th} solution is selected for division D_m
$c_{dx}(m, n, u, v)$	via costs of the selected solutions of neighboring divisions D_m, D_u
P_c	set of critical paths violating timing constraints
$t(p)$	current timing of path p
s_c, s_s	a segment of critical path p_c; a segment to switch with s_c
$l(s)$	layer on which segment s is assigned

long runtime overhead can be prevented. In Figure 8(b), the bottom circle indicates a set of candidates for the division where net N_3 is. For divisions with a significant number of solutions, we sort the solutions based on their internal via violation costs, and 20 solutions with the least violations are taken as final candidates for such a division.

With these generated solution candidates, we formulate our solution selection problem as ILP as in Equation (9). The notations are listed with details in Table II. Explicitly, ILP Equation (9) targets optimizing both the division's internal via violation costs and its external costs with neighboring divisions. Since much attention has been devoted to timing throughout candidate generation, only vias and via violations are counted as elements of costs in Equation (9a). For a solution n of division m, its corresponding cost, $c_d(m, n)$, is the sum of its internal vias and violations, both of them multiplied with weights; while considering vias on the boundary between two divisions m and u, as shown in the red points in Figure 8(b), we attempt all the combinations of different candidates belonging to these two divisions and calculate $c_{dx}(m, n, u, v)$ based on the via costs still. As the number of via violations is much fewer than the vias, we prefer to set the violation weight 10 times as high as the weight of vias. The constraint in Equation (9b) guarantees that only one solution can be selected for each division, D_m. Meanwhile, the constraints in Equations (9c) to (9e) limit that b_{mnuv} is the product of a_{mn} and a_{uv}, based on the condition that a_{mn} is binary from the constraint in Equation (9g). Its form is very similar to Equation (4), due to their essence of solving the same assignment problem, and thus we omit the very detailed description of this formulation.

$$\textbf{min} \sum_{m \in D(Nc)} \sum_{n \in Sol(m)} c_d(m, n) \cdot a_{mn} + \sum_{m,u \in Dx(Nc)} \sum_{n \in Sol(m)} \sum_{v \in Sol(u)} c_{dx}(m, n, u, v) \cdot b_{mnuv}, \tag{9a}$$

$$\textbf{s.t.} \sum_{n} a_{mn} = 1, \qquad \forall m \in D(Nc),\ n \in Sol(m), \tag{9b}$$

$$a_{mn} \geq b_{mnuv}, \qquad \forall (m, u) \in Dx(Nc),\ n, v \in Sol(m), Sol(u), \tag{9c}$$

$$a_{uv} \geq b_{mnuv}, \qquad \forall (m, u) \in Dx(Nc),\ n, v \in Sol(m), Sol(u), \tag{9d}$$

$$a_{mn} + a_{uv} \leq b_{mnuv} + 1, \qquad \forall (m, u) \in Dx(Nc),\ n, v \in Sol(m), Sol(u), \tag{9e}$$

$$b_{mnuv} \text{ is binary}, \qquad \forall (m, u) \in Dx(Nc),\ n, v \in Sol(m), Sol(u), \tag{9f}$$

$$a_{mn} \text{ is binary}, \qquad \forall m \in D(Nc),\ n \in Sol(m). \tag{9g}$$

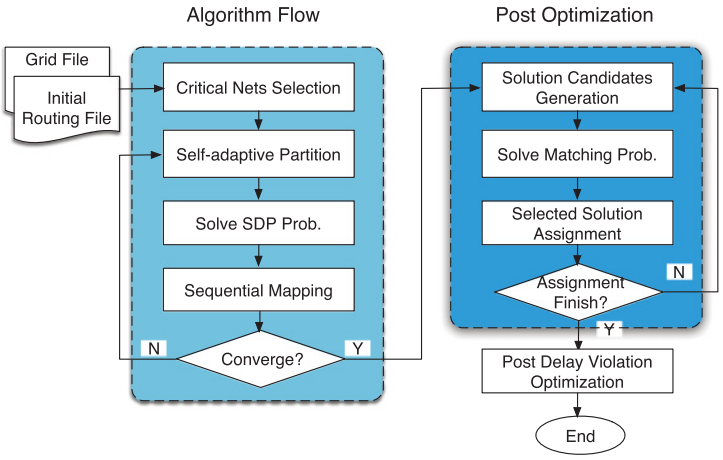

Fig. 9. Algorithm flow including matching algorithm.

To make this whole process more clear, we provide an overall algorithm flow including the proposed algorithm flow in Figure 9. After critical net selection and solving SDP, as shown in the left block, the iterative flow assigns segments through the sequential method to ensure convergence of SDP solutions. Considering that, if we integrate this concurrent algorithm into the same flow, the runtime overhead may be nonnegligible because of its ILP form. Therefore, we prefer to add this strategy as a post stage, as listed in the right block.

The flow in the right box takes the acquired SDP solutions as inputs. Based on these solutions, we generate more potential solution candidates to take more possibilities into account. Due to the massive number of divisions, we prefer to relax Equation (9) to iterative linear programming (LP) so that the promising solution can be selected for each division to form a whole assignment progressively. As the LP flow provides the noninteger solutions, we still set a threshold to determine its assignment. Here the threshold is set to 0.6. During each iteration, if the candidate's value exceeds 0.6, it is assigned to its corresponding division. Finally, to guarantee the completeness of the solution, when the number of residual divisions is small enough (i.e., smaller than 2,000), we prefer to resolve the rest divisions through the ILP and terminate this concurrent matching flow. In this manner, more solutions can be selected efficiently to avoid potential via violations and provide more reliable assignments.

3.6. Post Delay Optimization

The algorithm flow aforementioned gives a global view of timing optimization for critical path timing in nets, where these critical nets are selected as those with the most timing overheads. Considering existing timing constraints in practice, here we present a post sequential algorithm to reduce the timing violations as much as possible. As depicted beforehand, a net consists of one or more paths, where each path may lead to a possible timing violation. For those paths violating the specified constraint, they are symbolized as critical paths and endowed with priorities for high metal layers. With this premise, we present the details of our post greedy algorithm in Algorithm 2.

The outline of the post delay violation algorithm is listed in Algorithm 2, while the notations are also listed in Table II. As seen, a group of violating paths belonging to different critical nets are taken as the input to this algorithm. Different from ILP formulation, the post delay strategy targets reducing potential delay violations for a specified timing constraint. Thus, without overutilizing high layer resources for those

ALGORITHM 2: Post Delay Violation Algorithm

Input: A set of critical paths P_c;

```
1: Sort p with decreasing t(p_c);
2: for each p_c ∈ P_c do
3:     for each s_c ∈ p_c do
4:         for j = L_m; j ≥ 1; j = j − 2 do
5:             if j ≤ l(s_c) then
6:                 Break;
7:             end if
8:             Select s_s with the least vias;
9:             Switch l(s_c) and l(s_s);
10:            Update t(p_c) and t(p_s);
11:            if t(p_s) < T_0 then
12:                Break;
13:            else
14:                Restore l(s_c), l(s_s), t(p_c), t(p_s);
15:            end if
16:        end for
17:        if t(p_c) ≤ T_0 then
18:            Break;
19:        end if
20:    end for
21: end for
```

nets with large timing overheads, we allocate layer resources in a more balanced manner because these nets are no longer critical if they satisfy the timing constraint. This could provide more opportunities for those nets with slight violations. Notably, here we still focus on the maximum path timing of nets with violations, which complies with the ILP formulation. In our future work, we will consider timing paths consisting of multiple nets and cells simultaneously as an extension.

Since the clock frequency is generally affected by the path with the worst timing, we sort all the critical paths in decreasing order of their critical path timing (line 1). By starting from the path that has the worst violations compared to the given constraint, we traverse each critical segment, s_c, from its driver to sinks in a top-down manner, and search for higher metal layers to meet the delay constraint. With this objective, we search from the highest metal layer, the same as Algorithm 1, for a switching segment, s_s, which exists on a noncritical timing path.

To reduce its algorithmic complexity, instead of traversing all the segments on this edge, we prefer to choose one switching segment from a collection of segments sharing the edge with s_c. In this collection of segments, all their corresponding sink timing is smaller than 95% of the given constraint, T_0. Thus, we are able to avoid further delay violations induced by these reassigned segments. Before selecting s_s, as stated in lines 5 to 7, we check if the current layer is lower than the assigned layer of s_c. If so, we would turn to the next s_c on this path to reduce the timing overhead.

Then, during the selection of s_s, since all the segments with possible timing violations have been discarded before, we pay more attention to the resulting via costs. Thus, we prefer to select the segment with the least via costs as s_s. When there exists such an appropriate segment to switch with s_c, we exchange their assigned layers and update their path delays (lines 9–10). Before moving to the next s_c, we prefer to check the updated timing of path p_s and guarantee its legality (line 11). If p_s still satisfies the constraint, it is evident that a legal switching segment has been found for current s_c, and the rest of the lower layers can be skipped; otherwise, we should restore the previous assignment and seek for the next possible layer (line 14). To bound the surplus

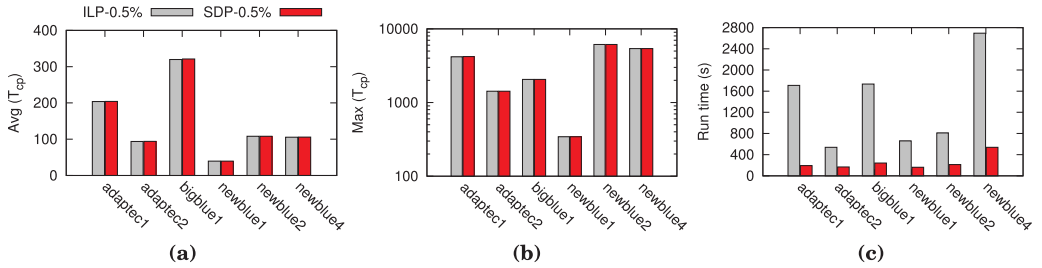

Fig. 10. Comparison between ILP and SDP on some small test cases. (a) On average delay for all critical paths. (b) On maximum delay for all critical paths. (c) On runtime.

search, we evaluate the updated timing of this critical path after each adjustment. If it meets the timing requirement, we freeze this path and continue to next critical path. Through this improvement, delay violations can be reduced explicitly with certain timing constraints.

4. EXPERIMENTAL RESULTS

4.1. Timing Results

The proposed layer assignment framework is implemented in C++ and tested on a 32-core Linux machine with 2.9GHz Intel® Core and 192GB memory. We select GUROBI [Gurobi Optimization Inc. 2016] as the ILP solver, and CSDP [Borchers 1999] as the SDP solver. Besides, we utilize OpenMP [Chandra 2001] for parallel computing and set the thread number to 16. As in Yu et al. [2015a], we test our framework on ISPD 2008 global routing benchmarks [Nam et al. 2008]. It should be noted in our experiments that both the resistance and capacitance values are from industrial settings, and thus our experimental results may have better agreement with industry timing.

In the first experiment, we compare the ILP formulation (see Section 3.1) with the SDP-based methodology (see Section 3.3 and Section 3.4). Since ILP formulation may suffer from the runtime overhead problem (i.e., it cannot finish in 2 hours for some large test cases), we select some small test cases for the comparison as shown in Figure 10.

Note that the partitioning technique is applied to both methods. We can see from Figure 10(a) and Figure 10(b) that SDP can obtain very similar average timing and maximum timing with ILP for these cases. This means that our SDP-based methodology provides an efficient relaxation with ILP formulation. Meanwhile, for these test cases, SDP can achieve significant speedup (see Figure 10(c)).

In the second experiment, we further evaluate our SDP-based method by comparing it with TILA [Yu et al. 2015a]. To make a fair comparison, we release the same set of nets for both TILA and our SDP. Table III lists the comparison results for the SDP-based method with TILA-0.5%. Here "0.5%" means 0.5% of most critical nets are released for both methodologies. Columns "Avg (T_{cp})" and "Max (T_{cp})" give the average and maximum timing of the critical path for all critical nets, respectively. Meanwhile, Columns "# of OV" and "# of via" list via capacity overflow and via count. The runtime is also reported in the Column "CPU(s)." From Table III, we can see that comparing with TILA, our SDP-based method can reduce the average timing by 11%, while the maximum timing can also be decreased by 4%. Since TILA also devotes efforts in maximum timing optimization, the improvement of maximum timing is reasonable. Our work also pays a slight penalty of via violations by 2% and keeps the same via count number as TILA. In addition, the reported runtime of SDP increases by 1.35 times in comparison with TILA, because the SDP problem is more complicated than min-cost flow problem. However, since we adopt the adaptive partitioning in the SDP-based

Table III. Performance Comparison on ISPD'08 Benchmarks

	TILA-0.5% [Yu et al. 2015a]					SDP-0.5%				
bench	Avg(T_{cp}) (10^3)	Max(T_{cp}) (10^3)	# of OV	# of via (10^5)	CPU(s) (s)	Avg(T_{cp}) (10^3)	Max(T_{cp}) (10^3)	# of OV	# of via (10^5)	CPU(s) (s)
adaptec1	228.57	4,378.42	49,205	19.31	132.9	204.88	4,205.71	50,947	19.26	112.5
adaptec2	97.94	1,435.79	38,173	19.25	133.8	93.88	1,421.68	38,480	19.32	91.2
adaptec3	220.00	4,613.89	90,961	36.74	322.5	209.41	4,583.29	92,299	36.76	569.0
adaptec4	121.67	5,616.23	72,695	32.22	272.4	117.43	5,590.84	73,185	32.44	494.3
adaptec5	249.51	5,406.11	81,151	55.21	444.8	216.15	5,311.75	84,537	55.26	472.0
bigblue1	402.81	2,673.18	44,399	21.69	174.7	322.41	2,065.42	46,256	21.56	142.1
bigblue2	100.94	10,821.67	114,343	43.38	188.7	95.58	10,728.23	115,240	43.49	264.9
bigblue3	27.38	789.61	65,718	52.62	333.5	21.53	373.80	66,795	52.92	547.7
bigblue4	37.98	3,779.11	95,348	109.94	747.1	33.56	3,750.95	97,148	110.37	804.3
newblue1	43.11	344.32	57,063	22.34	106.9	39.52	343.09	57,744	22.44	98.7
newblue2	110.76	6,171.37	35,994	28.97	151.4	107.85	6,130.09	35,566	29.25	146.4
newblue4	111.53	5,660.31	84,684	47.57	305.9	105.53	5,395.42	85,159	47.73	365.2
newblue5	170.45	2,789.52	152,770	86.65	605.1	151.41	2,771.55	157,944	87.00	1,564.4
newblue6	144.42	2,373.86	94,489	78.47	683.4	124.75	2,298.74	97,859	78.53	562.2
newblue7	30.03	1,301.30	143,087	163.81	1,161.2	25.33	1,254.22	144,580	164.28	1,555.7
average	139.81	3,896.98	81,339	54.54	384.3	124.61	3,748.32	82,916	54.71	519.4
ratio	**1.00**	**1.00**	1.00	1.00	1.00	**0.89**	**0.96**	1.02	1.00	1.35

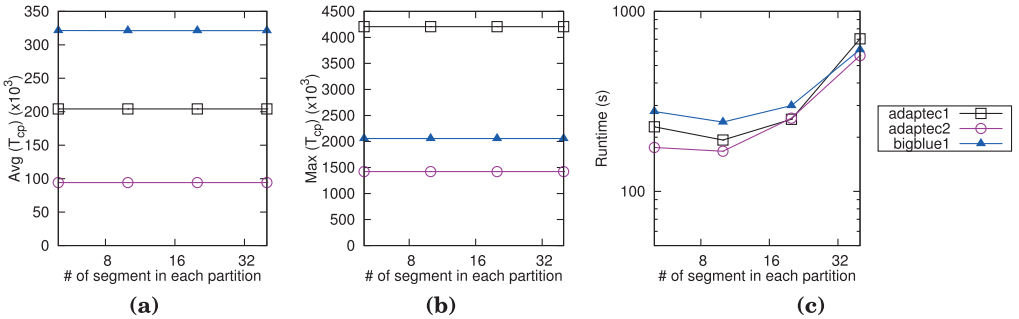

Fig. 11. Partition size impact on three small cases. (a) The impact on Avg(T_{cp}). (b) The impact on Max(T_{cp}). (b) The impact on runtime.

method (see Section 3.2), this method can still achieve reasonable runtime. During partitioning, we set its allowed number of segments in each partition as 10 for further self-adaptive partitioning methodology.

In the third experiment, we demonstrate the effectiveness of our self-adaptive partitioning methodology for SDP, as shown in Figure 11. We try different partition granularities (from five to 40) for three small test cases, where the maximum number of segments in each partition is limited. From Figure 11(a) and Figure 11(b), the average and maximum timing are quite similar, which means that partitioning has a negligible impact on performance because the tighter constraints would lead to more partitions. Although each partition is dealt in parallel with multiple threads, the impact of the performance is insignificant. Furthermore, Figure 11(c) shows that the runtime increases drastically with the partition granularity. Notably, without a self-adaptive partitioning methodology, the number of critical segments to deal with is so high that it takes around 1,000 seconds to run even a small benchmark by releasing 0.5%. Therefore, we can see that self-adaptive partitioning methodology benefits the runtime for SDP significantly. Meanwhile, we can observe that when the constraint is set to 10,

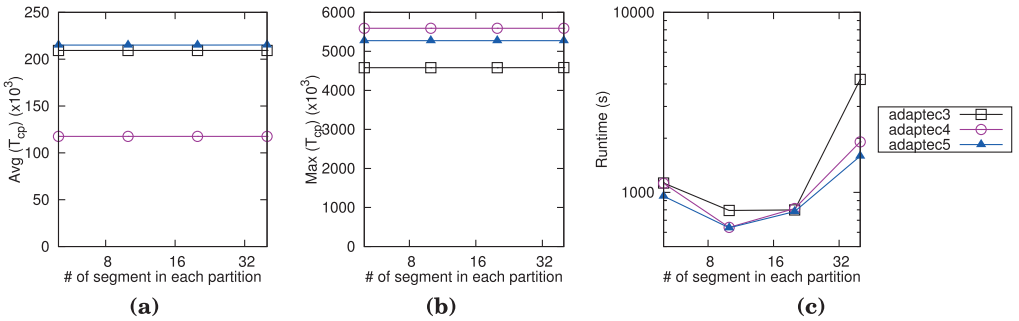

Fig. 12. Partition size impact on three large cases. (a) The impact on Avg(T_{cp}). (b) The impact on Max(T_{cp}). (b) The impact on runtime.

the runtime can reach its lowest point. Considering that small benchmarks may not be able to represent all the cases, here we also adopt the same set of experiments on three relatively large benchmarks, as shown in Figure 12. With the same selection of granularity, almost no difference is observed for the average and maximum timing results, respectively. Notably, the best runtime is still acquired when the granularity reaches 10 for each partition. One difference from small benchmarks is that the granularity of 20 segments in each partition is able to reach a similar runtime for benchmark "adaptec3," and 20-segment granularity can even spend less runtime than five-segment granularity, on average. The main reason is that, for larger benchmarks, a higher number of tasks are acquired from fine-grained partitions, which may lead to more runtime overheads. Thus, to balance the tradeoff between partition granularity and task number, a slightly larger partition granularity can be desired, because a fine-grained partition provides more opportunities for parallel execution, while a coarse-grained partition reduces the task number. But even for large benchmarks, the granularity of 10 segments can still achieve the lowest point while maintaining the similar performance of average and maximum timing. Therefore, in our implementation, we set the default partition granularity as 10.

In the fourth experiment, we evaluate the impact of thread number with the same partition granularity on three small benchmarks in Figure 13. As limited by machine resources, the thread number can be lower than 16 and higher than four; thus, we select four numbers, four, eight, 12, and 16, as the thread number for comparison. Observe that for both average and maximum timing results the performance differences are negligible, similar to the results shown in Figure 11 and Figure 12. In comparison, with the increase of threads, the runtime keeps decreasing until it reaches 16. It is seen that with the same problem size, a slight increase of threads will lead to obvious speedups, but when the number reaches a certain threshold, the speedup space will be limited due to the increasing communication overheads among various threads. Therefore, we set the number of threads in our framework to 16, which would lead to the most speedups.

To prove the effectiveness of our post delay optimization, we show the results by excluding this stage in the framework. Since a similar number of vias and via violations can be achieved, we provide the differences of maximum path timing and delay violations with and without the post optimization, as shown in Table IV. From Table IV, the maximum path timing of those selected critical nets can be improved slightly by 0.2%. Because we start to fix delay violations from the most critical net, for some designs the maximum path timing can be improved sufficiently; nevertheless, considering that both TILA and CPLA have placed adequate emphasis on that of critical nets, the further improving space has been constrained. Thus, this little improvement is reasonable. Also, we can observe that the number of delay violations can be well controlled with

Table IV. Performance Comparison With/Without Post Opt

	WO Post		W Post	
bench	Max(T_{cp}) (10^3)	# of Vio	Max(T_{cp}) (10^3)	# of Vio
adaptec1	4,205.71	6,215	4,205.71	5,474
adaptec2	1,421.68	5,148	1,421.68	3,820
adaptec3	4,583.44	9,812	4,583.40	8,367
adaptec4	5,591.10	11,091	5,591.10	8,429
adaptec5	5,311.75	21,273	5,273.78	17,583
bigblue1	2,065.42	12,871	2,056.57	11,329
bigblue2	10,728.70	18,211	10,725.35	16,004
bigblue3	373.80	5,122	304.20	1,149
bigblue4	3,751.01	10,451	3,751.06	2,294
newblue1	343.09	8,282	343.09	7,044
newblue2	6,130.09	11,065	6,130.09	6,961
newblue4	5,395.42	12,599	5,395.42	9,327
newblue5	2,771.74	40,471	2,771.74	32,196
newblue6	2,298.74	15,288	2,298.74	11,286
newblue7	1,254.22	13,823	1,254.22	4,555
average	3,748.39	13,448	3,740.41	9,721
ratio	**1.000**	**1.000**	**0.998**	**0.723**

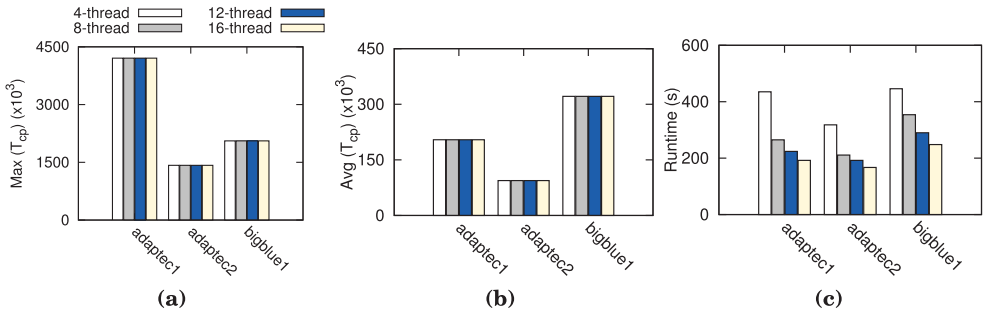

Fig. 13. Performance evaluation based on the number of threads on some small test cases. (a) Maximum delay for critical paths. (b) Average delay for critical paths. (c) Runtime.

the integration of post delay optimization. In our post optimization strategy, when a critical net satisfies the timing constraint, we will turn to the next violating one and this will leave more routing resources for the residual critical nets. In this way, this post optimization stage helps to fix 27.7% delay violations on average.

In the sixth experiment, we further analyze the impact of critical ratio to the performance of the SDP-based method. Critical ratio is an important parameter to determine how many critical nets are released. In Table III, we release 0.5% critical nets to see the improvement. Here we evaluate the SDP-based method by releasing more critical nets. Meanwhile, we compare the average critical path timing, maximum critical path timing, and runtime with TILA for one small benchmark adaptec1. From Figure 14(a) and Figure 14(b), we see that the average timing decreases slightly with the increase of critical ratio for both SDP and TILA. However, for maximum timing comparison, we see that TILA does not control the maximum timing well. The reason may be that TILA applies a Lagrangian-based relaxation optimization for via capacity constraints, which may affect the timing improvements. In Figure 14(c), we observe that for the

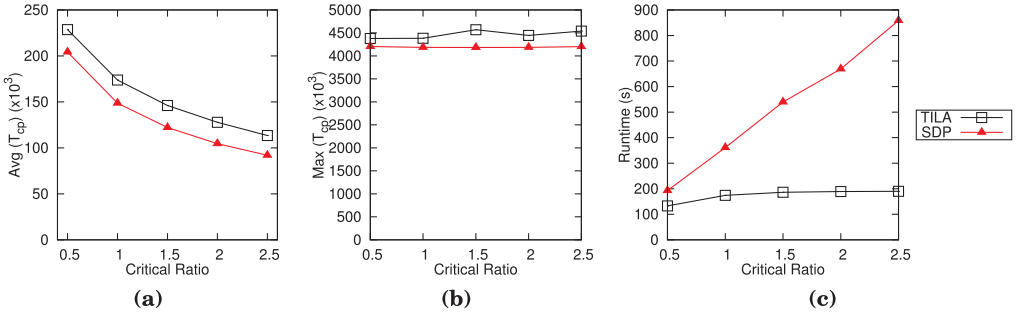

Fig. 14. Critical ratio impact on benchmark adaptec1. (a) The impact on Avg(T_{cp}). (b) The impact on Max(T_{cp}). (b) The impact on runtime.

SDP-based method the runtime increases in proportion to the critical ratio. This illustrates that our method has a well-controlled scalability.

4.2. Timing Violation Results

The third-last experiment manifests the effect of our post stage to reduce timing violations. To compare with CPLA results in Liu et al. [2016], we work on the same ISPD 2008 global routing benchmarks, which do not contain any delay constraint information. Considering the different sizes of test cases, we prefer to arrange appropriate criteria based on their initial timing conditions. Therefore, we set the delay constraint to 80% of the minimum timing of those critical nets in Table III. Table V lists the compared delay violations, also with the corresponding timing and via results.

From the results, it is shown that the overall number of timing violations is reduced by 28%. Meanwhile, both average and maximum critical path timing are very similar compared to CPLA. Considering the partial adjustment of the layers of segments on these paths, we can see the timing effect is quite obscure. Meanwhile, the number of via costs and violations is also well controlled. The only penalty we pay is runtime overhead, which increases by 43%. Due to the integration of concurrent mapping and post delay optimization, this runtime overhead is acceptable.

Additionally, we provide a comparison on via costs between sequential mapping and concurrent mapping, both of which occur in those divisions with one more candidate. The evaluation is tested on five benchmarks with different sizes to show its effectiveness. As seen from Figure 15(a), around 2% of via violations can be reduced; meanwhile, the number of via violations decreases by 0.34% in Figure 15(b). Therefore, under reasonable runtime overhead, the concurrent matching strategy is deserved.

Besides, to provide an explicit view of overflow distribution among layers, we measure the number of edge overflows on each layer and via violations between every two adjacent layers for NVM [Liu and Li 2011], TILA, and CPLA. As the initial input from NVM provides very few edge overflows, here we selectively pick three test cases with edge violations originally to show the violation distribution. Due to the existent control mechanism, both TILA and CPLA will not aggravate the initial edge violations throughout the whole procedure. Thus, they are able to keep the same edge violations as NVM, as shown in Figure 16(a), Figure 16(b), and Figure 16(c). Then, in the view of via violations, we observe that the majority of them occur in the bottom layers for all the results, although high metal layers have been utilized efficiently for timing optimization. This observation is also acceptable because ISPD 2008 global routing benchmarks provide much fewer tracks for lower metal layers than higher layers. Based on Equation (3), when there is no free track on a certain layer, via violations cannot be avoided anyway. From this perspective, via violations tend to appear on

Table V. Delay Violations Comparison on ISPD'08 Benchmarks

	CPLA in [Liu et al. 2016]						CPLA New					
bench	# of Vio	Avg(T_{cp}) (10^3)	Max(T_{cp}) (10^3)	# of OV	# of via (10^5)	CPU(s) (s)	# of Vio	Avg(T_{cp}) (10^3)	Max(T_{cp}) (10^3)	# of OV	# of via (10^5)	CPU(s) (s)
adaptec1	6215	204.88	4205.71	50947	19.26	127.4	5474	204.21	4205.71	50865	19.26	192.8
adaptec2	5147	93.88	1421.68	38480	19.32	105.2	3820	94.06	1421.68	38479	19.33	167.2
adaptec3	9812	209.41	4583.29	92299	36.76	569.0	8367	209.28	4583.40	92311	36.77	793.4
adaptec4	11089	117.43	5590.84	73185	32.44	494.3	8429	117.60	5591.10	73183	32.46	639.1
adaptec5	21269	216.15	5311.75	84537	55.26	472.0	17583	215.08	5273.78	84472	55.26	648.0
bigblue1	12852	322.41	2065.42	46256	21.56	150.4	11329	321.27	2056.57	46219	21.57	242.6
bigblue2	18215	95.58	10728.23	115240	43.49	264.9	16004	95.40	10725.35	115217	43.49	391.1
bigblue3	5122	21.53	373.80	66795	52.92	547.7	1149	21.92	304.20	66768	52.96	822.1
bigblue4	10434	33.56	3750.95	97148	110.37	804.3	2294	34.98	3751.06	97127	110.46	1272.2
newblue1	8296	39.52	343.09	57744	22.44	98.7	7044	39.51	343.09	57756	22.45	161.8
newblue2	11056	107.85	6130.09	35566	29.25	146.4	6961	108.03	6130.09	35552	29.26	213.2
newblue4	12599	105.53	5395.42	85159	47.73	365.2	9327	105.68	5395.42	85139	47.74	539.3
newblue5	40444	151.41	2771.55	157944	87.00	1564.4	32196	151.43	2771.74	157955	87.06	1978.5
newblue6	15273	124.75	2298.74	97859	78.53	562.2	11286	124.26	2298.74	97685	78.53	909.0
newblue7	13823	25.33	1254.22	144580	164.28	1555.7	4555	25.29	1254.22	144568	164.36	2166.1
average	13443	124.61	3748.32	82916	54.71	521.8	9721	124.53	3740.41	82886	54.73	742.4
ratio	**1.00**	**1.00**	**1.00**	1.00	1.00	1.00	**0.72**	**1.00**	**1.00**	1.00	1.00	1.43

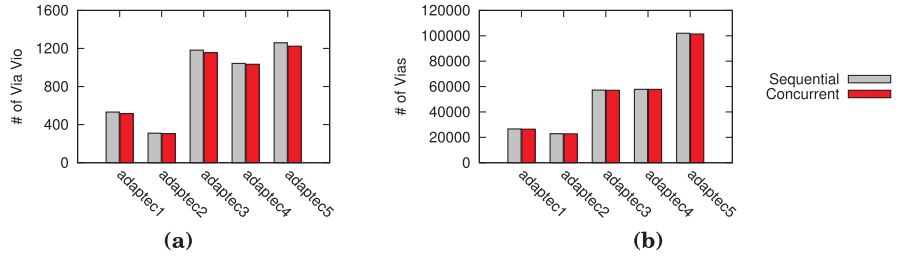

Fig. 15. Comparison between sequential mapping and concurrent matching. (a) Via violations. (b) Number of vias.

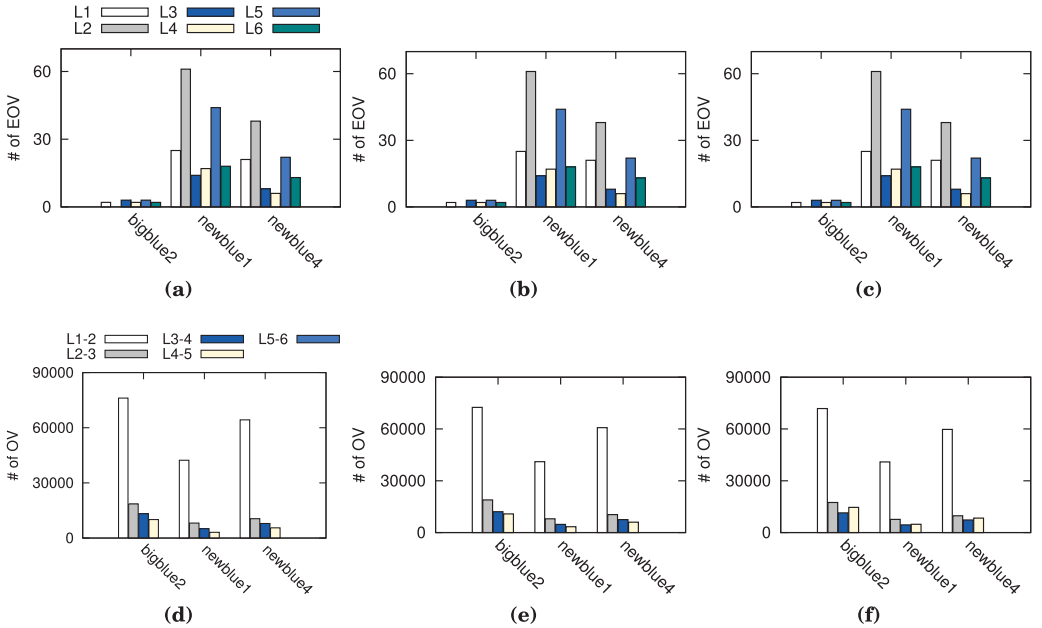

Fig. 16. Violation comparison of NVM, TILA, and CPLA. (a) Edge overflows in NVM;. (b) Edge overflows in TILA. (c) Edge overflows in CPLA. (d) Via overflows in NVM. (e) Via overflows in TILA. (f) Via overflows in CPLA.

layers with few available routing tracks. Meanwhile, we can also see that CPLA shows a similar distribution of via violations of TILA, based on the fact that they have a similar optimal objective. Additionally, the number of via violations on high layers in Figure 16(f) is slightly higher than that in Figure 16(e). This corresponds to the fact that high layers have been employed more sufficiently through CPLA for better timing achievement. Here we assume that via violations result only from stacked vias, so no violations exist on the lowest and highest layers.

5. CONCLUSION

This article targets optimizing critical path timing during the layer assignment stage. First, we propose the ILP formulation for the problem and then present the self-adaptive quadruple partition algorithm to benefit the runtime for SDP. Based on this speedup algorithm, the SDP-based method is developed. Additionally, an iterative LP framework is integrated as the post stage with another algorithm to reduce delay violations for paths. The experimental results show that our work can outperform

TILA by 11% for the average delay and 4% for the maximum delay of the critical paths. Besides, with the post delay optimization, timing violations can also be reduced efficiently. Since both TILA and CPLA focus on intranet delays, critical nets are selected based on their individual delays. As a future work, we plan to take timing paths into consideration that consist of multiple nets and cells together and coordinate the layer assignments of segments from different paths to meet the final constraint, with the consideration of transition vias affecting routing resources potentially.

REFERENCES

Jianchang Ao, Sheqin Dong, Song Chen, and Satoshi Goto. 2013. Delay-driven layer assignment in global routing under multi-tier interconnect structure. In *ACM International Symposium on Physical Design (ISPD'13)*. 101–107.

Brian Borchers. 1999. CSDP, A C library for semidefinite programming. *Optimization Methods and Software* 11 (1999), 613–623.

Rohit Chandra. 2001. *Parallel Programming in OpenMP*. Morgan Kaufmann.

James Hsueh-Chung Chen, Theodorus E. Standaert, Emre Alptekin, Terry A. Spooner, and Vamsi Paruchuri. 2014. Interconnect performance and scaling strategy at 7 nm node. In *IEEE International Interconnect Technology Conference (IITC'14)*. 93–96.

Jason Cong and Bin Liu. 2012. A metric for layout-friendly microarchitecture optimization in high-level synthesis. In *ACM/IEEE Design Automation Conference (DAC'12)*. 1239–1244.

Ke-Ren Dai, Wen-Hao Liu, and Yih-Lang Li. 2009. Efficient simulated evolution based rerouting and congestion-relaxed layer assignment on 3-D global routing. In *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC'09)*. 570–575.

Peng Du, Shih-Hung Weng, Xiang Hu, and Chung-Kuan Cheng. 2011. Power grid sizing via convex programming. In *International Conference on ASIC (ASICON'11)*. 337–340.

Ananda D. Gunawardena, S. K. Jain, and Larry Snyder. 1991. Modified iterative methods for consistent linear systems. *Linear Algebra Applications* 154 (1991), 123–143.

Gurobi Optimization Inc. 2016. Gurobi Optimizer Reference Manual. http://www.gurobi.com.

Chin-Hsiung Hsu, Huang-Yu Chen, and Yao-Wen Chang. 2008. Multi-layer global routing considering via and wire capacities. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD'08)*. 350–355.

Meng-Kai Hsu, Nitesh Katta, Homer Yen-Hung Lin, Keny Tzu-Hen Lin, King Ho Tam, and Ken Chung-Hsing Wang. 2014. Design and manufacturing process co-optimization in nano-technology. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD'14)*. 574–581.

Jiang Hu and Sachin S. Sapatnekar. 2001. A survey on multi-net global routing for integrated circuits. *Integration, the VLSI Journal* 31, 1 (2001), 1–49.

Andrew B. Kahng, Bao Liu, and Sheldon X.-D. Tan. 2006. Efficient decoupling capacitor planning via convex programming methods. In *ACM International Symposium on Physical Design (ISPD'06)*. 102–107.

Yukihide Kohira, Tomomi Matsui, Yoko Yokoyama, Chikaaki Kodama, Atsushi Takahashi, Shigeki Nojima, and Satoshi Tanaka. 2015. Fast mask assignment using positive semidefinite relaxation in LELECUT triple patterning lithography. In *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC'15)*. 665–670.

Tsung-Hsien Lee and Ting-Chi Wang. 2008. Congestion-constrained layer assignment for via minimization in global routing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 27, 9 (2008), 1643–1656.

Tsung-Hsien Lee and Ting-Chi Wang. 2010. Simultaneous antenna avoidance and via optimization in layer assignment of multi-layer global routing. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD'10)*. 312–318.

Derong Liu, Bei Yu, Salim Chowdhury, and David Z. Pan. 2016. Incremental layer assignment for critical path timing. In *ACM/IEEE Design Automation Conference (DAC'16)*. 85:1–85:6.

Derong Liu, Bei Yu, Salim Chowdhury, and David Z. Pan. 2017. TILA-S: Timing-driven incremental layer assignment avoiding slew violations. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* (2017).

Wen-Hao Liu and Yih-Lang Li. 2011. Negotiation-based layer assignment for via count and via overflow minimization. In *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC'11)*. 539–544.

Vinicius Livramento, Derong Liu, Salim Chowdhury, Bei Yu, Xiaoqing Xu, David Z. Pan, Jose Luis Guntzel, and Luiz C. V. dos Santos. 2017. Incremental layer assignment driven by an external signoff timing engine. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* (2017).

Gi-Joon Nam, Cliff Sze, and Mehmet Yildiz. 2008. The ISPD global routing benchmark suite. In *ACM International Symposium on Physical Design (ISPD'08)*. 156–159.

Maurice Queyranne. 1986. Performance ratio of polynomial heuristics for triangle inequality quadratic assignment problems. *Operations Research Letters* 4, 5 (1986), 231–234.

Daohang Shi, Edward Tashjian, and Azadeh Davoodi. 2016. Dynamic planning of local congestion from varying-size vias for global routing layer assignment. In *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC'16)*. 372–377.

Lieven Vandenberghe and Stephen Boyd. 1996. Semidefinite programming. *SIAM Review (SIREV)* 38, 1 (1996), 49–95.

Lieven Vandenberghe, Stephen Boyd, and Abbas El Gamal. 1997. Optimal wire and transistor sizing for circuits with non-tree topology. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD'97)*. 252–259.

Bei Yu, Derong Liu, Salim Chowdhury, and David Z. Pan. 2015a. TILA: Timing-driven incremental layer assignment. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD'15)*. 110–117.

Bei Yu, Kun Yuan, Duo Ding, and David Z. Pan. 2015b. Layout decomposition for triple patterning lithography. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 34, 3 (March 2015), 433–446.

Received October 2016; revised January 2017; accepted March 2017