The Dissertation Committee for Derong Liu
certifies that this is the approved version of the following dissertation:

Layer Assignment and Routing Optimization for Advanced Technologies

Committee:

David Z. Pan, Supervisor

Zhuo Li

Michael Orshansky

Nan Sun

Nur A. Touba

Layer Assignment and Routing Optimization for Advanced Technologies

by

Derong Liu

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN

August 2018

Acknowledgments

I am sincerely grateful to my adviser, Professor David Z. Pan, for his generous guidance and inestimable support along the way. Professor Pan introduced me to the magical research field of electronic design automation, where my interests was ignited. During the years, Professor Pan encourages me to ask questions, teaches me to consider problems carefully and guides me to solve the challenging problems with passion. In addition, Professor Pan has helped me to find several valuable internships, where my view has been broadened to gain more industrial experiences. His kind suggestions and support benefit me greatly in both research and general life. I am very fortunate to obtain guidance from Professor Pan in many aspects.

Also, I would like to express my appreciation to my committee members for offering me very valuable advice. In particular, I would like to thank Professor Michael Orshansky for the technical advice I acquired about this dissertation and several VLSI related courses. I would like to thank Professor Nan Sun for the constructive comments in the perspective of circuit design. I would also like to thank Professor Nur A. Touba for the helpful suggestions provided during the course and this dissertation. I would like to thank Dr. Zhuo Li for the great guidance, discussions, and support during my internships at Cadence.

I would express my great thanks to Dr. Salim Chowdhury who provided lots of industrial insights during our collaboration. This brought out significant benefits to my research. Meanwhile, I am also very thankful for my other mentors and colleagues during my internships: Dr. Charles Alpert, Dr. Jhih-Rong Gao, Dr. Wing-Kai Chow and Dr. Amin Farshidi at Cadence Design System, Dr. Duo Ding, Dr. Yue Xu, Mr. Huy Vo, Dr. Akshay Sharma, Dr. Yang Zhang, Dr. Xin Huang, Mr. Yu-Yen Mo, and Dr. Rajendran Panda at Oracle Corporation. Also, I am really thankful to Dr. Zheng Wang, Mr. Zhoufeng Ying, and Dr. Ray T. Chen for the kind suggestions about the optical project. Without these collaborations, this work would be impossible to accomplish.

I would like to thank all UTDA alumni and members, who provided lots of great help and suggestions for me all the time: First, I would like to thank Dr. Bei Yu for the very invaluable help regarding my research. Also, I would like to thank Dr. Subhendu Roy, Dr. Xiaoqing Xu, Dr. Yibo Lin, Dr. Jiaojiao Ou, Dr. Vinicius Livramento, Biying Xu, Zheng Zhao, Wei Ye, Jingyi Zhou, Meng Li, Wuxi Li, Shounak Dhar, Mohamed Baker Alawieh, Che-Lun Hsu, Keren Zhu, et al. With all these cherishable discussions and experiences during the years, I could set up the ideas and continue the research along the way.

I would express my sincere gratitude to my parents for their constant love. They encourage me to pursue my dream and support my decision with great belief. No words can express my appreciation. Also, I would like to send special thanks to Hai Yan, for his love and company during the years. Meanwhile, I also appreciate the help and support of all my friends.

Layer Assignment and Routing Optimization for Advanced Technologies

Publication No. ____________

Derong Liu, Ph.D.
The University of Texas at Austin, 2018

Supervisor: David Z. Pan

As VLSI technology scales to deep sub-micron and beyond, it becomes increasingly challenging to achieve timing closure for VLSI design. Since a complete design flow consists of several phases, such as logic synthesis, placement, and routing, interconnect synthesis plays an important role which includes buffer insertion/sizing and timing-driven routing. Although progress has been achieved by many advanced routing techniques, the following aspects can be exploited sufficiently for further improvement: (1) incremental layer assignment for timing optimization; (2) signal routing with the requirement of regularity; (3) power-efficient optical-electrical interconnect paradigm. Thus, to perform the layer assignment and routing optimization for advanced technologies, an automated routing engine in a global view is essential to benefit the interconnect design while satisfying specific requirements.

This dissertation proposes a set of algorithms and methodology on layer assignment and routing optimization for advanced technologies. The research includes two timing-driven incremental layer assignment approaches, synergistic topology generation and routing synthesis for signal groups, and optical-electrical routing design for power efficiency.

For incremental layer assignment, most of the conventional approaches target via minimization but neglect the timing issues. Meanwhile, via delays are ignored but should be considered in emerging technology nodes. Then two timing-driven incremental layer assignment frameworks are proposed, where all the nets are solved simultaneously with the integration of via delays: (1) optimization of the total sum of net delays and reduction of slew violations; (2) minimization of critical path timing in selected nets.

For on-chip signal routing, the bundled bits in one group may have different pin locations, but they have to be routed in a regular manner by sharing common topologies. Very few previous works target inter-bit regularity via multi-layer topology selection. Furthermore, the routability and wire-length of the signal bits should also be optimized. Then an advanced synergistic routing engine is promoted, which is able to not only control routability and wire-length but also guide each bit routing intelligently for design regularity.

For optical-electrical co-design routing, optical interconnect shows its advantage due to the dominance of bandwidth-distance-power properties. The previous works lack a detailed exploration of optical-electrical co-design for on-chip interconnects. During the transmission, signal quality can be affected by various loss sources and Electrical to Optical (EO)/Optical to Electrical (OE) conversion overheads should also be considered. Then a power-efficient routing flow for on-chip signals is presented, where optical connections can collaborate with electrical wires seamlessly.

The effectiveness of proposed algorithms and techniques is demonstrated in this dissertation. These approaches are able to achieve the improvements regarding specific metrics and eventually benefit the routing flow.

Table of Contents

List of Tables

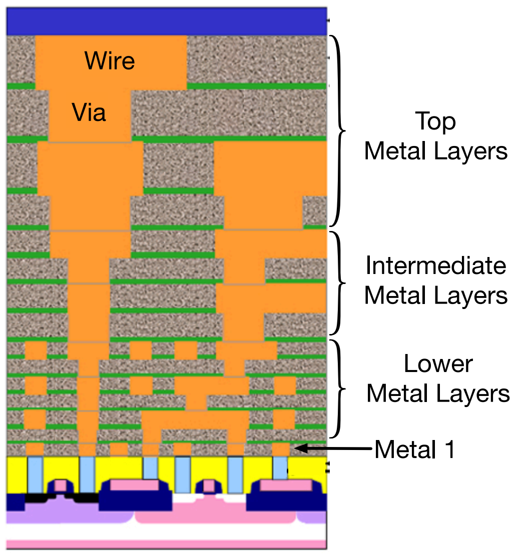
List of Figures

Chapter 1

Introduction

As VLSI technology scales to deep sub-micron and beyond, interconnect delay plays a determining role in timing [14]. Therefore, interconnect synthesis, including buffer insertion/sizing and timing-driven routing, becomes a critical problem for achieving timing closure [17]. 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 [13, 15, 29, 53, 62, 72, 84]. In emerging technology nodes, back-end-of-line (BEOL) metal stack offers heterogeneous routing resources, i.e., dense metal at the lower layers and wider pitches at the upper layers. Fig. 1.1 gives one example of cross section of IC interconnection stack in advanced technology nodes [1], 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 [30] are also listed. Advanced routing algorithms should not only be able to achieve routability, but also intelligently assign layers to overcome timing issues.

The current routing flow is given in Fig. 1.2, integrated with the proposed studies for layer assignment and routing optimization. Global routing provides the general routes aligned to the given grids, layer assignment places



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

Figure 1.1: The cross section of IC interconnection stack in advanced technology nodes [1], 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: [30]).

the routing wires onto appropriate layers, and detailed routing specifies the exact routes with the satisfaction of design rules, followed by a post-optimization stage for final refinement. As global routing and layer assignment play determining roles to routing results, first two layer assignment approaches are introduced, both targeting at timing optimization for all the nets simultaneously. Considering the limitations of layer assignment by adjusting topologies in one dimension, it is prominent to optimize the topologies in a broader view to gain more advantages. Thus, this dissertation provides a synergistic topology generation and route synthesis flow to direct the routing of signal groups globally. Besides the routing in the electrical field, with the advanced technologies, optical interconnect shows its competitiveness for signal commu-

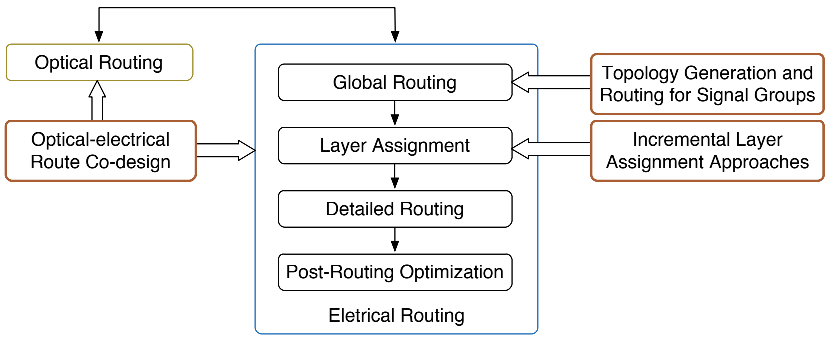

Optical Routing
Optical-electrical Route Co-design
Global Routing
Layer Assignment
Detailed Routing
Post-Routing Optimization
Eletrical Routing
Topology Generation and Routing for Signal Groups
Incremental Layer Assignment Approaches

Figure 1.2: Integration of proposed studies (orange blocks) into the routing flow for advanced technologies.

nication. Compared to copper wires, optical connections provide faster speed, higher bandwidth, and lower power consumptions. Hence optical and electrical characteristics are exploited to implement a power-efficient co-design routing engine.

As an important step in global routing, layer assignment is responsible for assigning net segments onto different layers. An intelligent assignment can benefit both interconnect delay and the number of required buffers [46]. In emerging technology nodes, wires on top layers are significantly less resistive than the bottom layers but should compete for much fewer available tracks. Thus it becomes increasingly challenging to perform a legal layer assignment with limited routing resources. Many previous works target at via minimization, which may easily assign wires onto the bottom layers with timing degradation [15, 43, 54]. Since via delay plays a non-negligible role in interconnect delay [50, 91], its impact should also be considered for timing optimization. With the increasing number of nets, a one-by-one strategy benefits the runtime but results in local optimality [7]. Based on these observations, it is motivated to propose efficient timing-driven incremental layer assignment frameworks for solving all the nets in a global view.

For on-chip performance-critical signal groups, besides the wire-length optimization, the routes are required to share equivalent topologies with concurrent bending points. One methodology is to regard one bundled group as a virtual net for routing [60,89]. By condensing multiple bits into one group, signal routing necessarily entails the routability degradation [39]. Then a global optimization is essential to allocate the feasible spaces and avoid congestion issues. Besides the global routing design, some detailed explorations are also operated for routing orientation and pin accessibilities [61]. In this dissertation, an advanced synergistic engine is provided to direct the signal routing considering routability, wire-length, and regularity in a 3D manner.

With the development of technologies, it brings about the feasibility to combine electrical wires with the optical paradigm for power efficiency. Due to the high bandwidth provided by optical interconnects, e.g. Wavelength Division Multiplexing (WDM), on-chip communication density increases efficiently [22]. Nevertheless, for the optical mechanism, the following issues have been noticed: (1) Optical-electrical conversion overheads result from optical modulators and detectors [23]; (2) Optical interconnect suffers from photon-energy loss from various sources [9]; (3) Sensitivity to process and thermal variation impacts the bit error rate (BER) for optical links [83]. By incorporating optical and electrical design, this dissertation proposes a power-efficient co-design routing engine.

This chapter first summarizes the current developments and difficulties regarding layer assignment and routing optimization. Then an overview of this dissertation is provided in terms of proposed algorithms according to the emerging challenges.

1.1 Challenges and Proposed Techniques in Layer Assignment and Routing for Advanced Technologies

Timing-driven Incremental Layer Assignment Avoiding Slew Violations As introduced, traditional layer assignment works mainly target at via minimization without appropriate consideration about timing. Due to the different timing requirements and capacity constraints, assigning all the segments onto high metal layers is not the best way to utilize limited metal resources. Also, the typical net-by-net strategy may lead to local optimality. Then chapter 2 introduces an incremental timing-driven layer assignment framework, with both delay and slew optimization of all the nets simultaneously. Multiprocessing with the partition method is also utilized to reach runtime speed-up. As an incremental approach, this layer assignment can smoothly work with either type of global router. The effectiveness of the proposed framework is verified through both academia and industrial benchmarks.

Incremental Layer Assignment for Timing Optimization For the timing-critical nets, the maximum path timing may lead to potential delay violations. Thus it should be optimized efficiently compared to the total sum of segment and via delays. By considering the via delay and via capacity constraints, the layer assignment problem, in essence, is a non-linear optimization problem. This stimulates the requirement of promoting an accurate formulation to solve the problem. Then chapter 3 proposes a novel incremental layer assignment approach based on Semidefinite Programming. This framework can improve the maximum path timing for the critical nets efficiently in comparison to the previous one. By integrating the post-optimization stage, it is able to further control via overheads and reduce delay violations greatly.

Synergistic Topology Generation and Route Synthesis for Signal Groups For the routing of on-chip signal groups, regular topologies with parallel connections are highly preferred to reduce inter-bit variability spread on silicon. Besides, in comparison to conventional two-pin bus routing, the different number of pins for the signal bits increases the complexity to control topology regularity. The required parallel routes from multiple signals bring challenges to the routability because of the congestion. Additionally, the source-to-sink distance deviation should also be considered to avoid signal degradation. Then chapter 4 presents the synergistic topology generation and route synthesis for on-chip performance-critical signal groups. Besides the improvement of wire-length and routability, it also develops regular topologies for the bundled signal bits. With the proposed metric of regularity ratio, the solutions are determined while satisfying the capacity constraints. The topologies are further revised through a post-optimization flow to encourage the routability and reduce source-to-sink distance violations. It is shown that the framework allocates the signal routes in a more balanced manner, compared to the manual designs from experienced industry designers.

Optical-electrical Power-efficient Route Synthesis For the optical-electrical route synthesis, optical interconnects are desired to collaborate with electrical counterparts smoothly. Generally, optical configurations are deployed for distant connections, which suffer from optical loss during the transmission. To make sure the light power can be detected at the receiving side, the overall loss should be in efficient control. With the satisfaction of detection constraints, how to distribute the interconnections onto optical and electrical layers is required to explore to save the power overheads. Then an optical-electrical power-efficient routing flow is developed in chapter 5. For the given signal bits, the hyper nets and pins are constructed, based on which the optical-electrical co-design route solutions are derived. Then the appropriate solution is assigned selectively and waveguides are exploited sufficiently for multi-channel routing. Compared to the existing optical router, power consumptions are saved through this optical-electrical co-design.

In conclusion, chapter 6 provides a summary of this dissertation and also discusses potential future research topics.

Chapter 2

Timing-Driven Incremental Layer Assignment Avoiding Slew Violations

2.1 Introduction

As an important step in global routing, layer assignment is responsible for assigning 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 [46]. 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 [14].

This chapter is based on the journal: Derong Liu, Bei Yu, Salim Chowdhury, and David Z. Pan. "TILA-S: Timing-driven incremental layer assignment avoiding slew violations." IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (2018). I am the main contributor in charge of problem formulation, algorithm development and experimental validations.

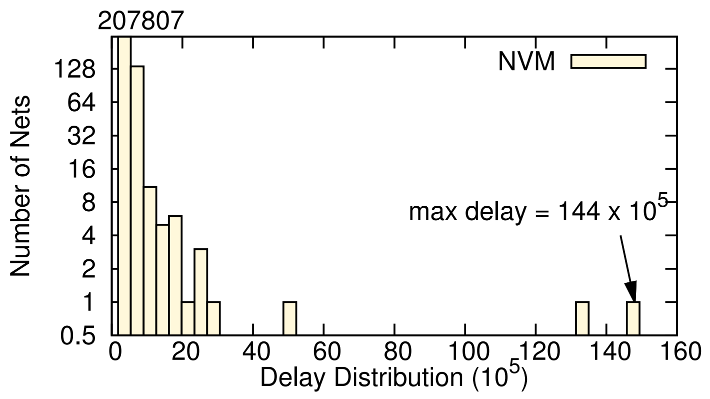
In addition, during timing closure slew violations could affect the utilization of buffering resources [31]. Thus to guarantee signal integrity and reduce buffering resources, slew violations need to be avoided during layer assignment.

Recently, layer assignment has been considered in two design stages, i.e., buffered tree planning and 3D global routing. Some studies consider layer assignment during buffer routing trees design [32,33,46]. Li et al. [46] proposed a set of heuristics for simultaneous buffer insertion and layer assignment. Hu et al. [32,33] proved that, even if buffer positions are determined, the layer assignment with timing constraints is $\mathcal{NP}$-complete. During 3D global routing, layer assignment is a popular technique for via minimization. Cho et al. [15] 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. [21, 44] 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 [7,54]. Ao et al. [7] 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 [45] to further reduce the number of vias. Furthermore, Lee et al. [43] proposed an enhanced global router with layer assignment refinement to reduce possible violations through a min-cost max-flow network. This framework works at one edge each time in a sequential order. For slew optimization, repeaters/buffers insertions are widely adopted to fix the potential slew violations [31, 46, 69].

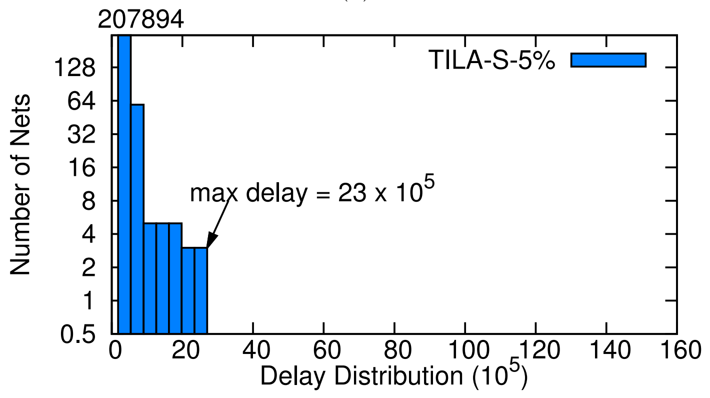
Zhang et al. [94] utilized an ILP approach to reconstruct the over-the-block steiner tree structure to improve slew.

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 the post-routing stage, slew violations may result in significant buffering resources. There are limited works to avoid slew violations globally during the layer assignment stage. (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 not 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 multi-net global optimization. For



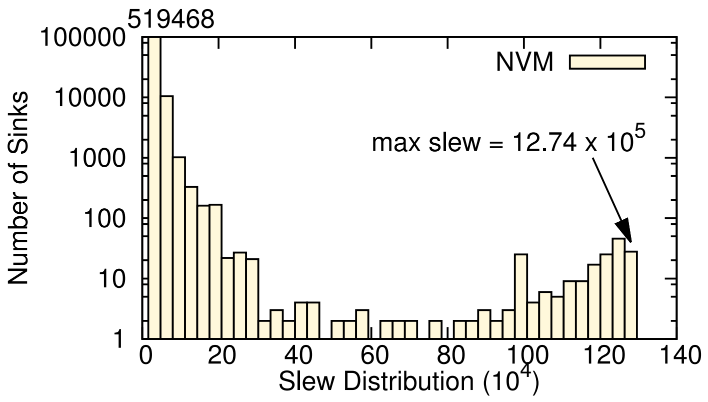
(a)



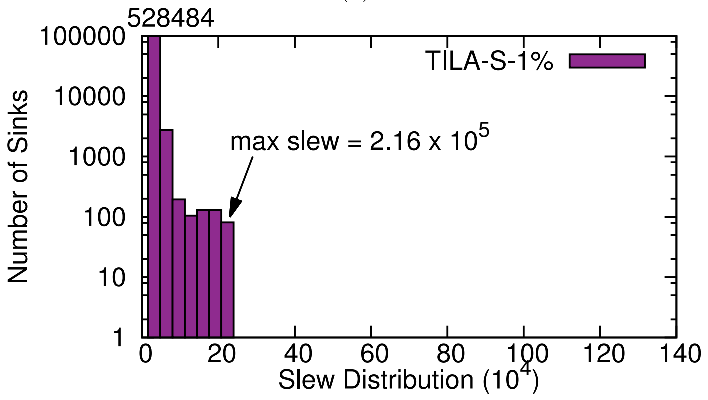
(b)

Figure 2.1: Net delay distribution for benchmark adaptec2. (a) Result by layer assignment solver NVM [54]; (b) Result by our timing-driven incremental layer assignment solver TILA-S, where 5% most critical nets are reassigned layers.

example, Fig. 2.1 compares the delay distributions of benchmark 'adaptec2' by conventional layer assignment solver [54] and the novel incremental timing-driven solution, while Fig. 2.2 compares the slew distribution results. It is seen that, since conventional layer assignment only targets at via minimization, the maximum delay and the maximum slew can be very large. Since the presented timing-driven planner is with a global view, the maximum delay can be much



(a)



(b)

Figure 2.2: Sink slew distribution for benchmark adaptec2. (a) Result by layer assignment solver NVM [54]; (b) Result by our timing-driven incremental layer assignment solver TILA-S, where 1% most critical nets are reassigned layers.

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.

This chapter proposes 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 [19]. Fast incremental improvements are developed in different timing optimization stages, such as incremental clock scheduling [6, 12], incremental buffer insertion [36], and incremental clock tree synthesis [74]. To further improve timing, incremental placement is also a very typical solution [57, 73]. Besides, there are several incremental routing studies (e.g. [93]) to introduce cheap and incremental topological reconstruction.

To the best of our knowledge, this work 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 [72, 84] or 2D routing followed by post-stage layer assignment [13,29,53,62]. Note that as an incremental layer assignment solution, this tool can smoothly work with either type of global router. The contributions are highlighted as follows.

- A mathematical formulation gives the layer assignment solutions with optimal total wire delays and via delays.
- A Lagrangian relaxation based optimization iteratively improves the layer assignment solution.
- A Lagrangian relaxation subproblem (LRS) is solved via a min-cost flow model that guarantees integer solutions due to the inherent uni-modular

property, thus, avoiding runtime extensive methods such as ILP.

- An iterative Lagrangian relaxation based slew optimization strategy is proposed to reduce the violations globally.
- A post slew optimization algorithm searches potentially usable layers for fixing local violations.
- Multiprocessing of $K \times K$ partitions of the whole chip provides runtime speed up.
- Both ISPD 2008 and industrial benchmarks demonstrate the effectivenesses of our framework.

The remainder of this chapter is organized as follows. Section 2.2 provides some preliminaries and the problem formulation. Section 2.3 gives the mathematical formulation and also proposes the sequence of multi-threaded min-cost flow algorithm to achieve further speed-up. In addition, how to mitigate slew violations is discussed in this Section. Section 2.4 reports experimental results, followed by the summary in Section 2.5.

2.2 Preliminaries and Problem Formulation

This section introduces the graph model and the timing model applied in this work. Then the problem formulation of timing-driven incremental layer assignment is provided.

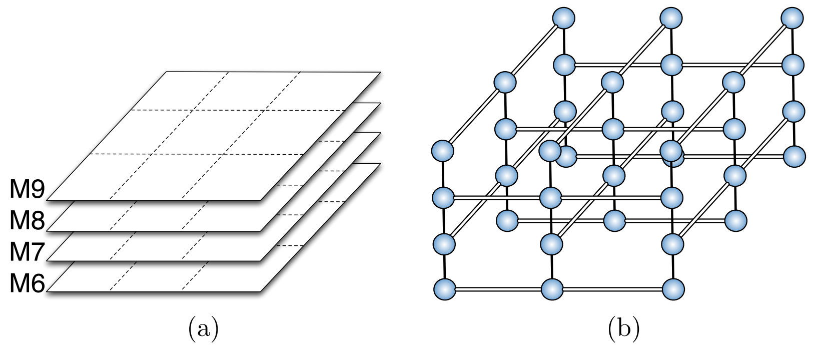


Figure 2.3: Layer design and grid models. (a) A design with four routing layers {M6, M7, M8, M9}; (b) Grid model with preferred routing directions.

2.2.1 Graph Model

Similar to the 3D global routing problem, the layer assignment problem can be modeled on a 3D 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. 2.3(a) shows a grid graph for routing a circuit in a multi-metal layer manufacturing process. Each metal layer is dedicated to either horizontal or vertical wires. The corresponding 3D grid graph is shown in Fig. 2.3(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 [28]. 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. If there is no routing space for those two edges, no vias are allowed to be inserted in this cell. 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.

2.2.2 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. 2.4, 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:

$$d_e(i,l) = R_e(l) \cdot (C(l)/2 + C_{down}(s_i)), \tag{2.1}$$

where $R_e(l), C(l)$ refer to the edge resistance on layer l, and edge capacitance on layer l, respectively. $C_{down}(s_i)$ 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 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

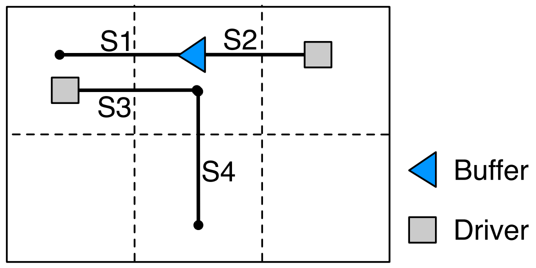


Figure 2.4: Example of net model.

can be calculated as follows.

$$d_v(v_m, l) = R_v(l) \cdot C_{down}(v_m). \tag{2.2}$$

Here $R_v(l)$ is the resistance of via between layers l and $l+1$, and $C_{down}(v_m)$ 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, buffer positions can be considered in our delay model. That is, for one segment s_i, if there is one buffer at its end point, its downstream capacitance $C_{down}(s_i)$ should be equal to the buffer input capacitance. As shown in Fig. 2.4, $C_{down}(s_2)$ 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 in our assumption would affect the downstream capacitance for the corresponding pin. Meanwhile, integration with buffers 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 our work. Therefore, through updating the downstream capacitances and including buffer intrinsic delay, our framework can handle timing optimization for both pre-buffered and post-buffered designs.

2.2.3 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 [37]. The calculation is given in Eq. (2.3), where $Slw(p_u(s_i)), Slw(p_d(s_i))$ are the input and output slew of s_i, respectively, while $Slw_{step}(s_i)$ is the step slew.

$$Slw(p_d(s_i)) = \sqrt{Slw(p_u(s_i))^2 + Slw_{step}(s_i)^2}. \tag{2.3}$$

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 Eq. (2.3). Regarding the step slew, we calculate it through the combination of *PERI* model and *Bakoglu's metric*. It is proved to have an error within 4% [37]. The calculation is shown in Eq. (2.4), 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.

$$Slw_{step}(s_i) = Slw_{step}(i, l(s_i)) = ln9 \cdot d_e(i, l(s_i)). \tag{2.4}$$

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.

$$Slw_e(i, l(s_i)) = \sqrt{Slw(p_u(s_i))^2 + (ln9 \cdot d_e(i, l(s_i)))^2}. \tag{2.5}$$

Besides, via slew should also be considered during slew calculation and computed in a similar way as segment slew. Eq. (2.6) gives the slew for via v_m from layer l to layer $l+1$.

$$Slw_v(v_m, l+1) = \sqrt{Slw(p_{v_m})^2 + (ln9 \cdot d_v(v_m, l))^2}. \tag{2.6}$$

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.

2.2.4 Problem Formulation

Based on the grid model and timing model discussed in the preceding section, the timing-driven incremental layer assignment (TILA) problem is defined as follows:

Problem 1 (TILA) *Given a global routing grid, a set of critical net segments and layer / via capacity information, timing-driven incremental layer assignment 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.*

It shall be noted that in this work we only consider layer assignment for timing optimization, while other techniques such as buffering are not discussed. One instance of TILA problem with three nets is demonstrated in Fig. 2.5, where nets n_1 and n_2 are non-critical nets, while net n_3 is timing critical. In the initial layer assignment, net n_3 is assigned on 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. proved that even if no timing is considered, the decision version of layer assignment for via minimization is $\mathcal{NP}$-complete [64]. Thus the decision version of TILA problem is $\mathcal{NP}$-complete as well.

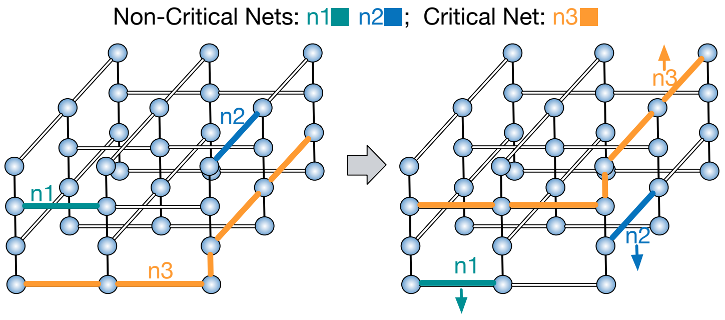


Figure 2.5: An example of timing driven layer assignment. In initial layer assignment net n_3 is timing critical. Through resource releasing from nets n_1 and n_2, the total timing gets improvement.

2.3 TILA-S Algorithms

In this section, we introduce our framework to solve the $\mathsf{TILA-S}$ 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 2.1.

2.3.1 Mathematical Formulation

The starting mathematical formulation of TILA problem is shown in Formula (2.7). In the objective function, the first term is to calculate the cost of segments, while the second term is to calculate the cost from vias. Here $d_e(i,j)$ is calculated through Eq. (2.1), and $d_v(i,p,k)$ is derived from Eq. (2.2).

Table 2.1: Notations for timing-driven layer assignment.

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. s_i's 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, where segment s_i passes on layer j
$c_g(k)$	available via capacity of g-cell g on layer k

Constraint (2.7b) 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 (2.7c) is for the edge capacity of each layer. Constraint (2.7d) 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 $C_{down}(s_i)$ is constant, the TILA can be formulated as an integer linear programming (ILP), then a mature ILP solver

$$\begin{aligned}
\min \quad & \sum_{i\in S}\sum_{j=1}^{L} d_e(i,j)\cdot a_{ij} + \sum_{(i,p)\in E_x}\sum_{j=1}^{L}\sum_{q=1}^{L}\sum_{k=\min(j,q)}^{\max(j,q)-1} d_v(i,p,k)\cdot a_{ij}\cdot a_{pq}, && \text{(2.7a)}\\
\text{s.t.} \quad & \sum_{j} a_{ij} = 1, \forall i \in [1,S], && \text{(2.7b)}\\
& \sum_{s_i\in S_e(i)} a_{ij} \le c_e(i,j), \forall e \in E, \forall j \in [1,L], && \text{(2.7c)}\\
& \sum_{(i,p)\in E_x(g)}\ \sum_{\min(j,q)\le k<\max(j,q)} a_{ij}\cdot a_{pq} \le c_g(k), \forall g\in G, \forall k \in (1,L), && \text{(2.7d)}\\
& a_{ij} \text{ is binary }. && \text{(2.7e)}
\end{aligned}$$

is possible to be applied. Here $C_{down}(s_i)$ is the downstream capacitance of segment s_i. We can use a boolean variable $\gamma_{ij,pq}$ to replace each non-linear term $a_{ij}\cdot a_{pq}$. Then Formula (2.7) can be transferred into ILP through introducing the following artificial constraints:

$$\begin{cases} a_{ij} + a_{pq} \le \gamma_{ij,pq} + 1, \\ a_{ij} \ge \gamma_{ij,pq}, \quad a_{pq} \ge \gamma_{ij,pq}. \end{cases} \tag{2.8}$$

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 (2.7e). 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 a reasonable good solution. Instead of expensive ILP formulation or its LP relaxation, our framework proposes a Lagrangian relaxation based algorithm to solve the original Formula (2.7).

2.3.2 Lagrangian Relaxation based Optimization

Lagrangian relaxation [75] is a solution technique for solving optimization problems with difficult constraints, where some or all hard constraints are moved into the 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 (2.7d) 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 the objective function. The modified formula is called Lagrangian relaxation subproblem (LRS), as shown in Formula (2.9). Through this relaxation methodology, via capacity overflow is handled with timing optimization simultaneously.

$$\begin{aligned}
\mathbf{min}\ & \sum_{i \in S} \sum_{j=1}^{L} d_e(i,j) \cdot a_{ij} \\
& + \sum_{(i,p) \in E_x} \sum_{j=1}^{L} \sum_{q=1}^{L} \sum_{k=\min(j,q)}^{\max(j,q)-1} d_v(i,p,k) \cdot a_{ij} \cdot a_{pq} \\
& + \sum_{(i,p) \in E_x} \lambda_{ij,pq} (a_{ij} \cdot a_{pq} - c_g(k)), \qquad (2.9) \\
\mathbf{s.t.}\ & (2.7\text{b}) - (2.7\text{c}), (2.7\text{e}).
\end{aligned}$$

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 Formula (2.7) [75]. That is, the original formulation is the primal problem and the Lagrange multiplier 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 TILA

```
Input: Initial layer assignment solution;
Input: Critical net ratio $\alpha$;
1: Select all segments based on $\alpha$;                ▷ Section 2.3.4
2: Initialize $C_{down}(s_i)$ for each segment $s_i$;
3: Initialize LMs;
4: while not converged do
5:     Solve LRS;                                         ▷ Section 2.3.3
6:     Update $C_{down}(s_i)$ for all $s_i$;
7:     Update LMs;
8: end while
```

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 non-critical nets (line 1). All the segments belonging to these (selected critical and non-critical) nets are reassigned layers by our incremental framework. Please refer to Section 2.3.4 for more details of our critical and non-critical net selection. Based on the initial layer assignment solution, we initialize all the $C_{down}(s_i)$ 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 Lagrangian relaxation subproblems (LRS), with fixed LM values (lines 4–8). In solving LRS, we minimize the objective function in Eq. (2.9) based on the current set of LMs. The details of solving LRS are discussed in Section 2.3.3. After solving each LRS, we re-calculate the downstream capacitances of all the segments $C_{down}(s_i)$ based on Eq. (2.1) (line 6). We use a subgradient-based algorithm [5] 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.

$$\lambda_{i,j,p,q} = \lambda'_{i,j,p,q} + \theta_{ijpq} \cdot (a_{ij} \cdot a_{pq} - c_g). \tag{2.10}$$

The available via resources can be obtained directly by updating the current via capacity as in [28]. To decide the step length, we adopt the classic calculation as follows:

$$\theta_{ijpq} = \frac{\phi \cdot [UB - L(\lambda_{i,j,p,q})]}{\|(a_{ij} \cdot a_{pq} - c_g)\|^2}. \tag{2.11}$$

Based on Eq. (2.11), 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 pre-specified fraction.

2.3.3 Solving Lagrangian Subproblem (LRS)

Through removing the constant items and reorganizing objective function of Formula (2.9), we re-write LRS into Formula (2.12).

$$\begin{aligned} \textbf{min} \; & \sum_{i=1}^{S}\sum_{j=1}^{L} c(i,j)\cdot a_{i,j} + \sum_{(i,p)\in E_x}\sum_{j=1}^{L}\sum_{q=1}^{L} c(i,j,p,q)\cdot a_{ij}\cdot a_{pq}, \qquad (2.12)\\ \textbf{s.t.} \; & (2.7\text{b}) - (2.7\text{c}), (2.7\text{e}), \end{aligned}$$

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

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

Due to the space limit, the detailed proof is omitted. Because of the nonlinear term $a_{ij}\cdot a_{pq}$, the proof can be acquired through a reduction from quadratic assignment problem [59]. In addition, unless $\mathcal{P}=\mathcal{NP}$, the quadratic assignment problem cannot be approximated in polynomial time within some finite approximation ratio [71]. Inspired by MacCormick Envelops, we prefer to linearize the term $a_{ij}\cdot a_{pq}$:

$$c(i,j,p,q)\cdot a_{ij}\cdot a_{pq} \approx c(i,j,p,q)\cdot (a'_{pq}\cdot a_{ij} + a'_{ij}\cdot a_{pq}), \qquad (2.13)$$

where a'_{pq} is the value of a_{pq} in the previous iteration, and a'_{ij} is the value of a_{ij} in the 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 Eq. (2.13). 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 Eq. (2.13), the objective function in Formula (2.12) 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 (2.7b) and (2.7c) can be integrated into the flow model through specified edge capacity. Constraint (2.7e) is satisfied due to the inherent uni-modular property of min-cost network flow [5].

An example of such a min-cost flow model is illustrated in Fig. 2.6. 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: start vertex s, segment vertices V_S, layer vertices V_L, and end vertex t. Here both the 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: $\{s \rightarrow V_S\}$, $\{V_S \rightarrow V_L\}$, and $\{V_L \rightarrow t\}$. 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 the corresponding layer; the costs of all other edges are set to 0. For segments whose directions are not compatible with certain layers, no edge exists between those segment and layer vertices. 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 the 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. 2.6, segment s_1 can be assigned on either layer 6 or layer 8 of edge 1; similarly, segment s_2 can also be assigned on two layers of edge 2. The numbers shown in V_L vertices indicate the specified layer of this edge and the corresponding edge index, respectively. The corresponding grid model is given in Fig. 2.4, where we can see that segment s_1 shares the same routing edge with s_3, therefore s_1 competes for the routing resource with s_3. Meanwhile, segment s_4 has a different routing

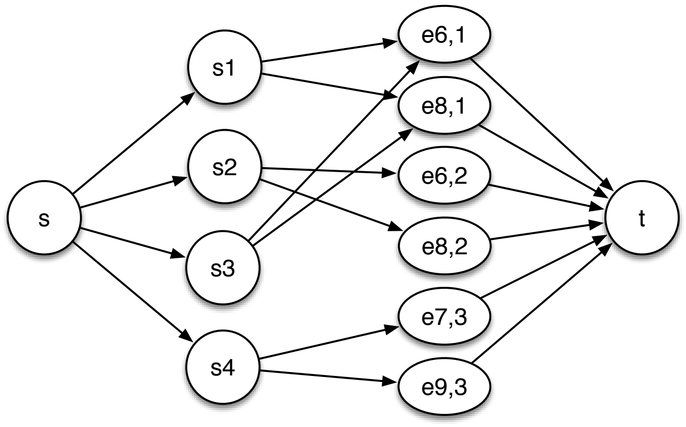


Figure 2.6: An example of min-cost flow model.

direction with the other three segments so it has to be assigned on other layers for vertical routing. When the number of segments to be assigned on one edge exceeds the edge routing capacity, our framework will assign the segments in order to minimize the assigning costs. In this example, we assume that each segment passes through one edge with its length equal to the grid size, as shown in Fig. 2.4. For a segment passing through multiple edges, we prefer to split it into a set of sub-segments, and each sub-segment has the same length as the grid size. We construct the flow graph where each sub-segment has its own assigning cost, and the number of sub-segments to be assigned on one layer is also constrained by the layer node.

2.3.4 Critical & Non-Critical 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 non-critical nets. Both the selected critical nets and the selected non-critical nets would be reassigned layers. The motivation of critical net selection is to reassign their layers to improve timing, while the motivation of non-critical net selection is to release some high layer resources to the critical nets. In this way, our incremental layer assignment flow is able to overcome the limitation of any net order in original layer assignment. In our implementation, the default value of α is set to 1, which means 1% of nets would be identified as critical nets, while the other 1% of nets are selected as non-critical nets.

To identify all the critical nets can be trivial: first, all the net timing costs in original layer assignment are calculated based on our delay model as in Section 2.2, and then the $\alpha\%$ of worst delays are selected. Yet, the non-critical net selection is not so straightforward, as randomly selecting $\alpha\%$ of best timing nets may not be beneficial to improve critical net timing. Therefore, we prefer to select those nets with the best timing sharing more routing resources with the critical nets while these nets are assigned on high metal layers. Otherwise, releasing the non-critical nets on lower layers has no benefits for final timing results. In our implementation, we check the $2 \cdot \alpha$ nets with the best timing and 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 a half of them with the best scores as the non-critical nets.

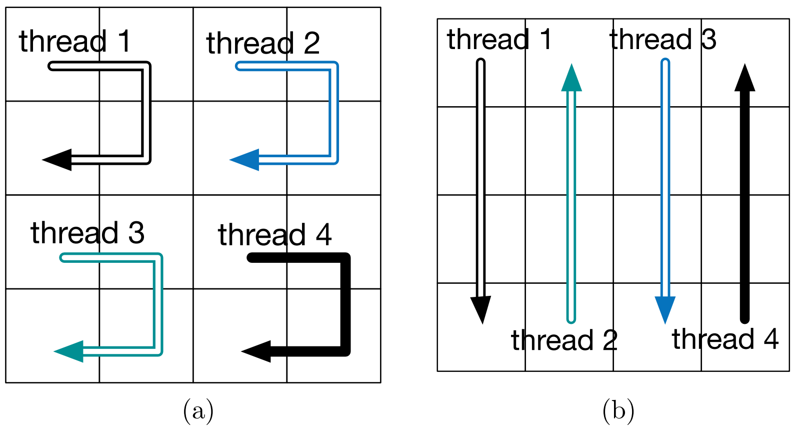


Figure 2.7: Our parallel scheme to support multi-threading computing on $K \times K$ partitions. (Here $K = 4$). (a) Parallel pattern 1; (b) Parallel pattern 2.

2.3.5 Parallel Scheme

Our framework supports the parallel scheme by dividing the global routing graph into $K \times K$ parts. An example of such a division is illustrated in Fig. 2.7, where $K = 4$. The timing-driven incremental layer assignment is solved in each partition separately. During partitioning, each segment is ensured to be solved in one and only one partition. To achieve this, for segments crossing boundaries between different partitions, they are assigned in the same partition as its geometric center. If its geometric center is exactly on the boundary, we assume this segment belongs to the partition in its left/bottom side. The reason for such a division is twofold. Firstly, our Lagrangian relaxation based optimization is to solve a set of min-cost flow models, as discussed in Section 2.3.2 and Section 2.3.3. The runtime complexity to solve a single flow model is $\mathcal{O}(|V| \cdot |E|)$, where $|V|$ and $|E|$ are the vertex number and

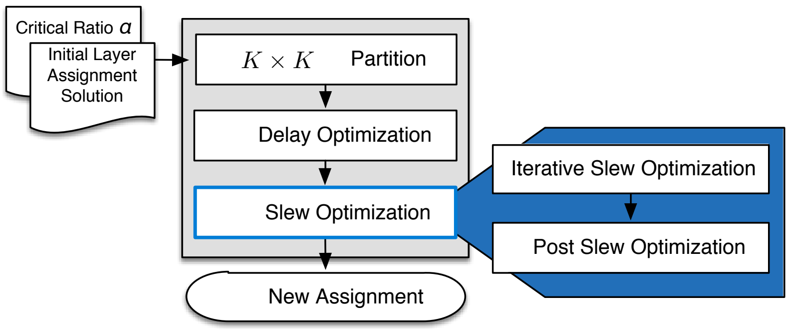


Figure 2.8: Overall timing optimization flow.

the edge number of the graph. Dividing the whole problem into a set of sub-problems can achieve significant speed-up. In addition, multi-threading is applied to provide further speed-up. For instance, in Fig. 2.7(a) four threads are used to solve different regions simultaneously. Secondly, inspired by the Gauss-Seidel method [25], when one thread is solving the 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 the above example, we also propose a more general type of parallel pattern suitable for any $K \times K$ partition, as illustrated in Fig. 2.7(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. This second pattern is more suitable for multi-processing considering its synchronization mechanism.

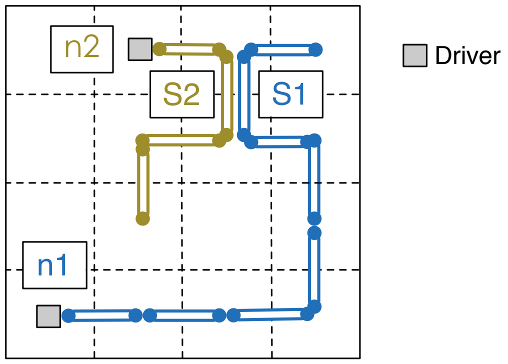


Figure 2.9: An example of difference between delay and slew optimization.

2.3.6 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 the number of slew violations besides delay optimization. Fig. 2.8 depicts the overall algorithm flow, which mainly consists of two stages: delay optimization and slew optimization. The details of delay optimization are already introduced from Section 2.3.2 to Section 2.3.5. As discussed in Section 2.2.3, segment step slew is in proportion to its delay. With the constant segment 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. 2.9. 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 higher layer resource. Regarding delay optimization, segment s_2 is possible to be assigned on a higher layer because it owes a larger downstream capacitance with a closer distance to its driver; while in fact, segment s_1 should be placed on a higher layer because it is on a longer path which may introduce slew violations. Through slew optimization flow as shown in Fig. 2.8, segment s_1 will be assigned a higher priority on a higher layer. The details of the algorithm flow will be given later. The main reason is that slew optimization considers the impact of both upstream segments and downstream segments. In this manner, slew optimization has a different impact on the assignment of critical nets in comparison to delay optimization. If we consider both optimizations simultaneously, they may affect each other to degrade the final performance. The detailed reasons are two-fold: First, critical nets can be selected in a different way during the delay and slew optimization. In the stage of slew improvement, these nets exceeding slew constraints are to be selected as critical nets to fix their violations; 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. Secondly, delay improvement targets at total delay reduction considering via overflows, while slew improvement targets at the reduction of slew violations. Due to different optimal objectives, assigning costs for both delay and slew optimization may lead to a trade-off based on their weights. Considering the assigning

Table 2.2: 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

differences of s_1 and s_2 in Fig. 2.9, 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 in an explicit manner, and reduce slew violations globally as a second stage after delay optimization.

Fig. 2.8 also outlines the slew optimization flow, whose input is the assignment result after delay optimization. The slew optimization consists of two steps: iterative slew optimization and post greedy optimization. This section focuses on the first step to reduce slew violations based on the flow model, while Section 2.3.7 provides the details of post slew optimization. Some notations used in the slew optimization are listed in TABLE 2.2.

In the iterative optimization, similar to delay optimization flow, the same ratio of critical and non-critical nets are selected based on their slews. 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 given slew constraint, this path is defined as a critical path, i.e. $P_{critical}$, and the exceptional slew is counted as the critical value. Meanwhile, segment input slews are initialized based on the input result because each segment should be reassigned simultaneously. Then we reassign these nets through iteration-based Lagrangian relaxation 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 a 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. Eq. (2.14) gives the slew constraint:

$$Slw(p_d(s_i)) \leq 0.95 \cdot Slw_c, \qquad i \in P_{critical}, \tag{2.14}$$

where $Slw(p_d(s_i))$ is the segment output slew, and Slw_c is the slew constraint. To solve this problem, we relax Eq. (2.14) through Lagrangian Relaxation by moving the slew calculation into the objective function, and eliminate all the $0.95 \cdot Slw_c$ because they are constants. Eq. (2.15) provides the corresponding 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.

$$\begin{aligned} \mathbf{min} \quad & \sum_{i \in P_{critical}} \sum_{j=1}^{L} \beta_{ij} \cdot Slw_e(i,j) \cdot a_{ij}, \qquad (2.15) \\ \mathbf{s.t.} \quad & (2.7\text{b}) - (2.7\text{e}). \end{aligned}$$

During each iteration, LMs are updated as shown in Eq. (2.16),

$$\beta_{ij} = \beta'_{ij} \cdot \sqrt{\frac{Slw_{sink}(P_{critical})}{Slw_c}}, \qquad (2.16)$$

where β'_{ij} is the LM in the previous iteration, and $Slw_{sink}(P_{critical})$ 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. 2.9, segment s_1 has a higher priority than s_2.

Similar to Eq. (2.7), Eq. (2.15) is solvable through ILP because we can obtain $Slw_e(i,j)$ based on the last iteration. Still, we incorporate the via capacity constraints into the objective function with the same linearization method as in Eq. (5.5). Ultimately, the problem can be formulated as a weighted sum of a_{ij}s and solved through the 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 the slew optimization flow terminates. In summary, this algorithm provides a slew targeted optimization because it considers both the upstream segments and downstream segments. Meanwhile, more emphasis is placed on critical paths by taking the sink slew into accounts.

Based on the slew model, the segment input slew can affect the output slew directly, but during each iteration, we obtain the input slew of each segment based on the last iteration. Thus, it may introduce slew discrepancies by calculating the segment slew based on the previous assignments. Therefore, we implement a post slew optimization algorithm, which mainly focuses on fixing local violations while considering current layer assignments of the whole path. The details of this algorithm are given in Section 2.3.7.

2.3.7 Post Slew Optimization

In this section, a post slew optimization algorithm is proposed to further reduce the slew violations. The pseudocode is shown in Algorithm 2. Based on the global optimization results, we traverse each net sink 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 5–34).

First, if there exists 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

Algorithm 2 Post Slew Optimization Algorithm

```
Input: 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 non-critical $s_p$ 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 $Slw_{sink}(P_{critical}) \leq Slw_c || Slw(i, l') \geq Slw_c$ then
30:            break;
31:        end if
32:    end for
33: end for
```

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 to improve the segment slew violation (lines 14–28).

In the second step, we search for a non-critical 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 non-critical 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 layer of segment s_i and segment s_p (line 19). If the improvement outperforms the current most 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 the corresponding nets (lines 26–27). When the slew violation of $P_{critical}$ has been fixed, then we continue to fix the next net in N_{slw}. The segments of each net are traversed in a top-down manner from driver to sinks. When 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 further reduce the runtime overhead. The algorithm ends until all nets in N_{slw} are traversed. In comparison to slew optimization

Table 2.3: Normalized capacitance and resistance.

Wire [30]			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	-	-

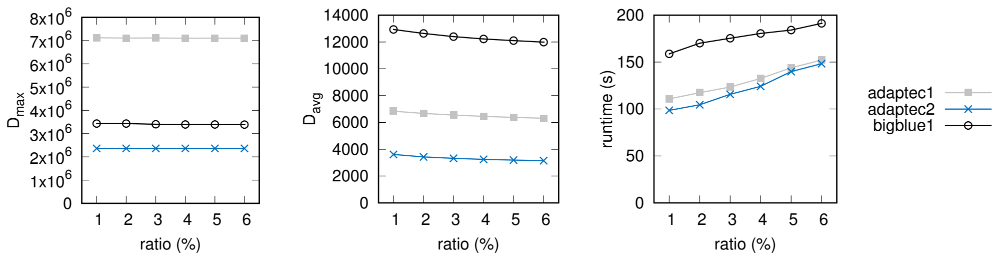
in Section 2.3.6, this algorithm adjusts the layer assignment of segments based on their real input slew, thus providing a more accurate slew optimization. Meanwhile, if there are only a few slew critical nets, it is efficient to fix the violations through this algorithm.

2.4 Experimental Results

We implemented the proposed timing-driven incremental layer assignment 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 [2] as our min-cost network flow solver, and utilized OpenMP [3] to provide parallel computing. In our implementation, the default K value is set to 6, and the default thread number is set to 6.

Table 2.4: Performance comparisons on ISPD 2008 benchmarks.

Bench	NVM [54]						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	–



(a) (b) (c)

Figure 2.10: Performance impact on different ratio values. (a) The impact of ratio on maximum delay; (b) The impact of ratio on average delay; (c) The impact of ratio on runtime.

2.4.1 Evaluation on ISPD 2008 Benchmarks

In the first experiment, we evaluate our timing-driven layer assignment framework on ISPD 2008 benchmarks [65]. The NCTU-GR 2.0 [53] is utilized to generate the initial global routing solutions. The initial layer assignment results are from NVM [54], which is targeting at via and overflow minimization. Our framework is tested the effectiveness to incrementally optimize the timing.

To calculate the wire delay in Eq. (2.1) and via delay in Eq. (3.2), all the metal wire resistances, metal wire capacitances, and via resistances are listed in TABLE 2.3. 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 [30], 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 2.4 compares NVM [54] 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 non-critical nets are reassigned layers. In TILA-5%, 5% of timing critical nets and 5% of non-critical nets are reassigned layers. For each methodology, columns "OE#", "OV#", "D_{avg}", "D_{max}", and "via#" list the resulting edge overflow, via overflow, average delay, maximum delay, and the total number of vias, separately. Here the calculation of via overflow is described in [28]. 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 [53] cannot generate a legal global routing solution where the number of segments passing one edge in 2-D dimension exceeds the total edge capacities. We also cannot report the results from another recent work [7], as for this benchmark suite their binary gets assertion fault before dumping out results.

From TABLE 2.4 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 number of overflows and the average delay are reduced by 3% and 10%, respectively. The penalty for such timing improvement is that the number of vias is increased by only 3%. On the average, TILA-1% requires around 409 seconds for each test case. Compared with extreme fast net-by-net solver NVM, although our planner solves a global optimization problem, its runtimes are reasonable. For instance, based on [54], for test cases `adaptec1` and `adaptec5`, NVM needs around 36 and 99 seconds, respectively. Our planner needs around 125 and 493 seconds, respectively. In TILA-5%, when 5% of the most critical nets are reassigned layers, the maximum delay is reduced by 53%. Meanwhile, the number of overflows and the average delay are reduced by 3% and 19%, respectively. The penalty of TILA-5% is that the number of vias is increased by 11%. From TABLE 2.4 we can see that even small amounts of critical nets (e.g. 1%) are considered, the maximum delay can be effectively optimized. When more nets are inputted in our planner, 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%.

Table 2.5: Performance comparisons on 20nm industrial 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**	-

Critical net ratio α is a user-defined parameter to control how many nets are released to incremental layer assignment. In TABLE 2.4, ratio α is set to 1% and 5%. Fig. 2.10 analyzes the impact of ratio value to the performance of incremental layer assignment framework. Fig. 2.10(a) shows the impact of ratio value on the maximum delay, where we can see that the maximum delays are kept the same. This means for these test cases, releasing 1% of critical nets is enough for maximum delay optimization. Fig. 2.10(b) shows the impact of ratio value on the average delay, where we can see increasing the ratio value can slightly improve the average delay. Fig. 2.10(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 trade-off between average delay and the speed of our tool.

Our incremental layer assignment utilizes OpenMP [3] to implement multi-threading. Fig. 2.11 analyzes the performance of our layer assignment framework under different partition and thread numbers. Thread 1 corre-

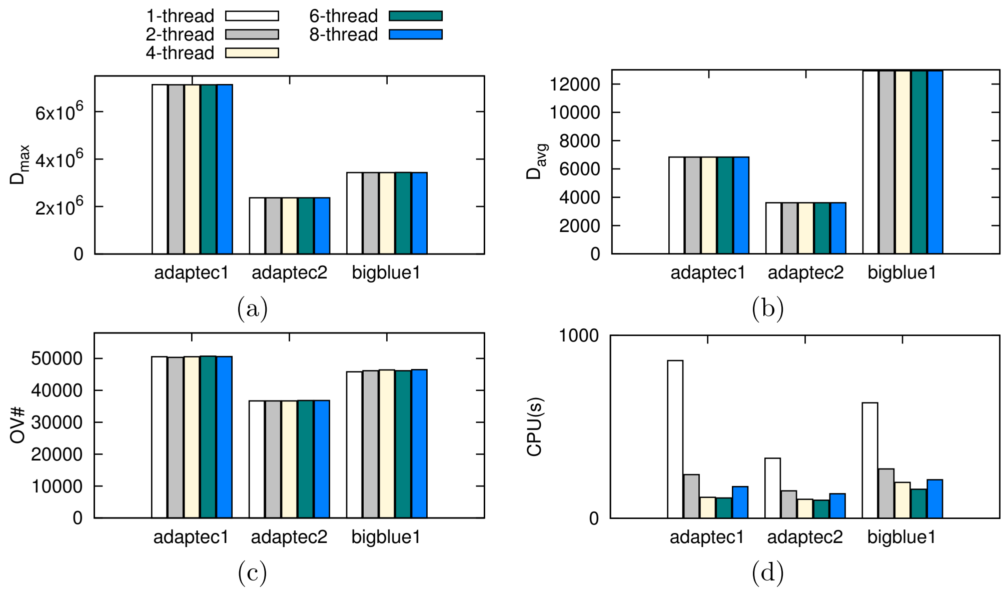


Figure 2.11: Evaluation thread number impact on three test cases in ISPD 2008 benchmark suite. (a) The impact on maximum delay; (b) The impact on average delay; (c) The impact on overflow; (d) The impact on runtime.

sponds to 1×1 partition, thread 2 corresponds to 2×2 partitions, and so on. With more partitions, the size of the network flow model is reduced quadratically thus benefiting the runtime significantly together with multi-threads. From Fig. 2.11(a) and Fig. 2.11(b) we can see that the impact of thread number on both maximum delay and average delay is insignificant. Similarly, through Fig. 2.11(c) we can see the impact on overflow is also negligible. From Fig. 2.11(d) 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,

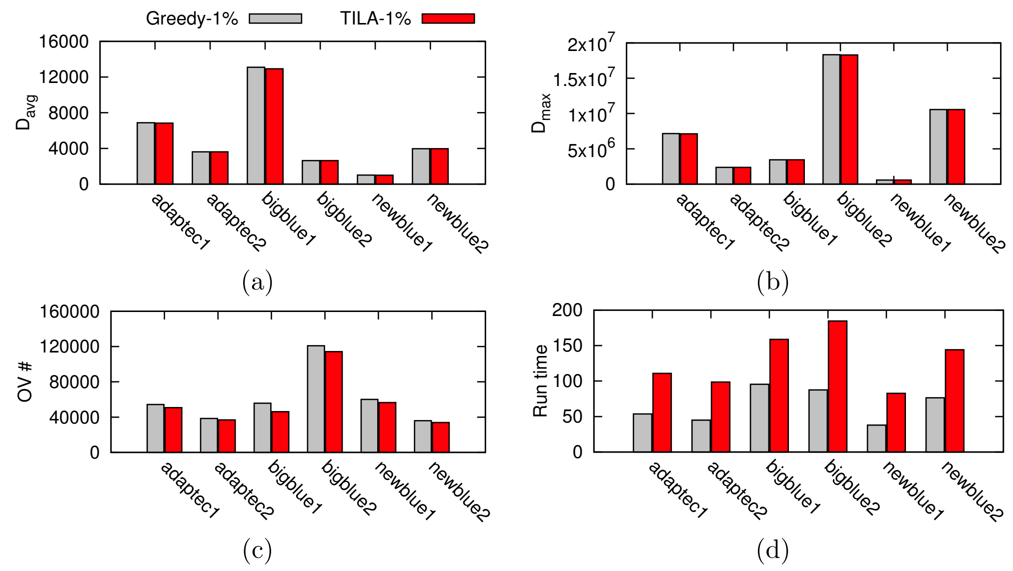


Figure 2.12: Comparison between greedy methodology and TILA on some small test cases: (a) on average delay; (b) on maximum delay; (c) on via overflow; (d) on runtime.

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 higher priority for higher metal resources. For each net, segments are traversed sequentially and layers are selected based on the same costs as that in the min-cost max-flow network. Here we release 1% critical nets and 1% non-critical nets. The results are shown in Fig. 2.12. From the figure, we can observe that for both average and maximum delay TILA can achieve a little bit better results compared with the greedy method. The main reason is that the greedy methodology assigns higher priorities to those critical nets so that these nets are able to take advantage of higher layer resources. Since those nets utilize high metal layers efficiently, significant timing optimization can also be achieved through this greedy methodology. Nevertheless, they sacrifice the via capacity violations due to their preferences for high layer resources. Regarding the runtime, as shown in Fig. 2.12(d), due to the net-by-net scheme, the greedy method is faster than TILA. Therefore, to control timing optimization and capacity constraint in a reasonable manner, a global optimization engine is more promising.

2.4.2 Evaluation on 20nm Industry Benchmarks

In the second experiment, we test our incremental layer assignment framework on eight 20nm 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 2.5 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 all the critical nets are provided in the benchmarks, the critical and non-critical 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 improvement and 34% average delay improvement. The total number of vias after our iterative optimization is very similar to the initial solution. The reasons to reach a similar number, or even a slightly better number of vias are due to the following factors: Firstly, critical segments are assigned

Table 2.6: Comparisons on ISPD 2008 benchmarks for slew optimization.

Bench	NVM [54]	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

on high metal layers while non-critical segments are assigned on low layers together with their neighboring segments. Few vias will be induced for those connecting segments are on close layers. Secondly, via delays are also included in our mathematical formulation, which also helps to control the via counts. Finally, industrial benchmarks provide a more even layer assignment of segments through all the layers. This provides us with a potential space for via counts optimization. The initial layer assignment solution is with zero overflow, and our framework can also maintain such zero overflow performance. In summary, from TABLE 2.5 we can see our incremental layer assignment framework can achieve significant timing improvement.

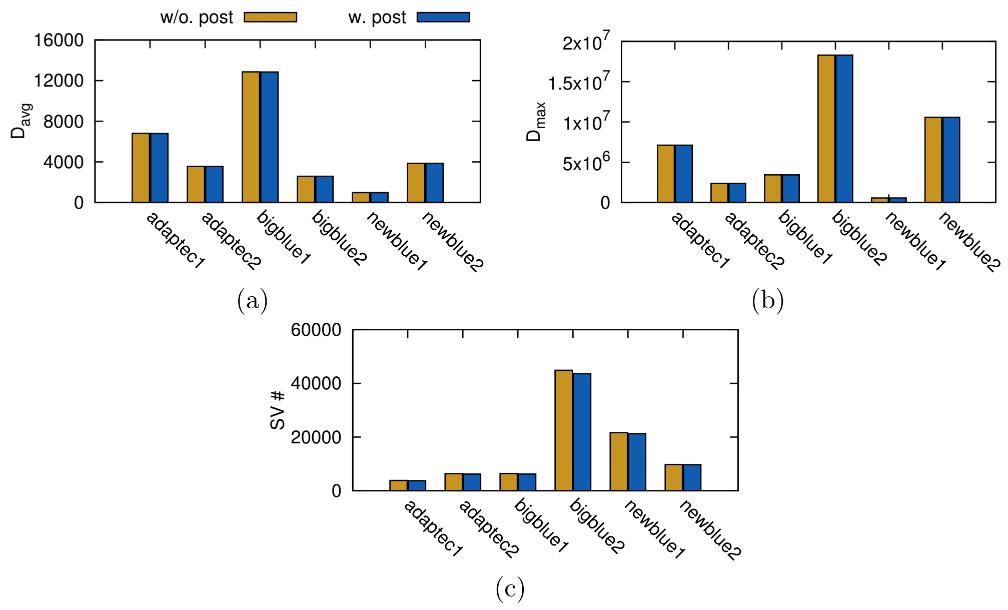


Figure 2.13: Comparison between with and without post slew optimization stage on some small test cases: (a) on average delay; (b) on maximum delay; (c) on slew violations.

2.4.3 Slew Comparisons on ISPD & 20nm Industry Benchmarks

In this section, we compare TILA with slew optimization (TILA-S) against TILA without slew improvement (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 5 times its initial average delay as shown in TABLE 2.4. In this manner, the initial number of slew violations is in proportion to the number of total segments for each benchmark. However, the slew constraints for industry benchmarks are given based on industrial settings.

TABLE 2.6 lists the results for ISPD benchmarks by comparing TILA-S-1% with TILA-1% while releasing 1%. Besides the performance metrics shown in TABLE 2.4, 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, the 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. 2.13 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 the selection of switching candidate segments, we take its current slew into consideration. Once the candidate is selected with the smallest slew degradation, its impact

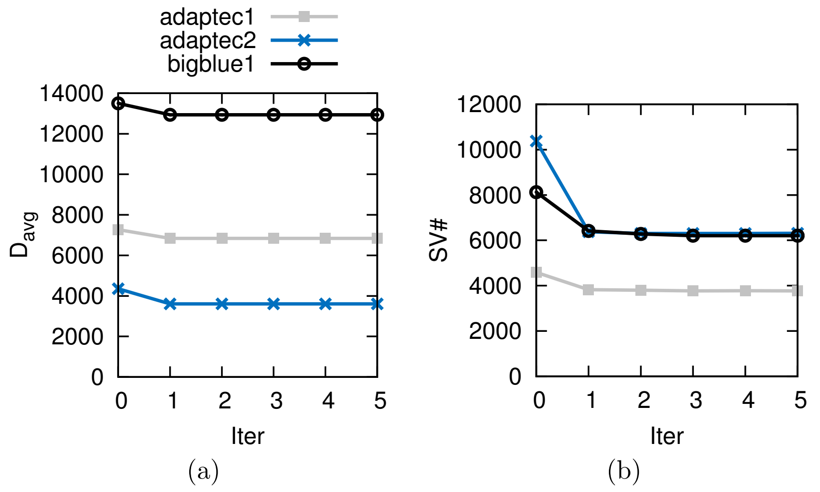


Figure 2.14: Convergence with iteration number of TILA-S on some small test cases: (a) on average delay; (b) on slew violations.

on delay is also negligible because slew is closely related with delay.

To illustrate the timing convergence of our iterative framework, we relax the convergence constraint for the delay and slew optimization, and record the average delay and slew violation number for each iteration till the fifth iteration. Fig. 2.14 shows the timing convergence with iteration number. The 0-th iteration corresponds to the initial solution, where we can see a clear convergence after the first two iterations.

As stated in Section 2.3.6 and Section 2.3.7, our slew optimization flow reduces the number of slew violations and benefits the buffering overhead. To make this explicit, we measure the number of buffers we may adopt for each ISPD 2008 benchmark in Fig. 2.15. Here we implement 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 one buffer when there is a slew violation;

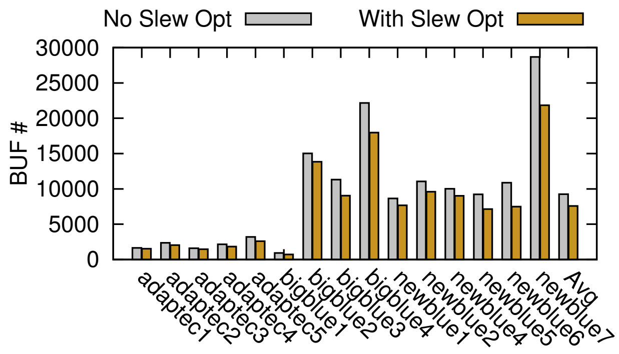


Figure 2.15: Buffering overhead saving with slew optimization.

Table 2.7: Comparisons on $20nm$ 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**

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 overhead can be reduced from 9258 to 7586 in Fig. 2.15. Therefore, our post slew-targeted optimization helps to reduce the buffering overhead, and is also able to provide an estimate of buffering overhead at the pre-buffering stage.

For the $20nm$ industry benchmarks, besides delay and via metrics, we also take slew violations into account. TABLE 2.7 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 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 numbers of slew violations in industrial benchmarks, we prefer to skip the first global optimal stage. The results from TABLE 2.7 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.

2.5 Summary

This chapter includes 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 the Lagrangian relaxation-based method is proposed to iteratively improve the timing of all the nets. The Lagrangian relaxation subproblem (LRS) is modeled through the 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 industrial benchmark suites, and has demonstrated its effectiveness.

Chapter 3

Incremental Layer Assignment for Timing Optimization

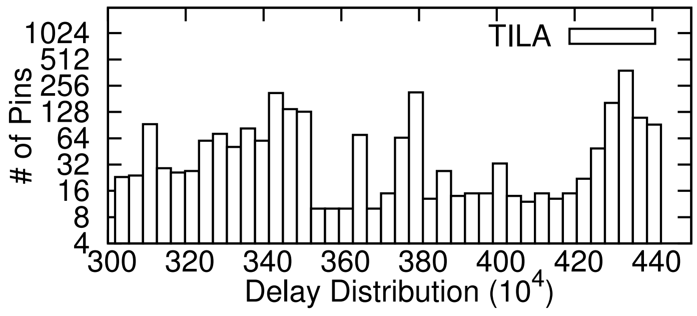
3.1 Introduction

The previous chapter has introduced one timing-driven incremental layer assignment framework, targeting the optimization of the total segment and via delays. To conquer its shortcomings, this chapter describes a novel incremental layer assignment for timing optimization of critical paths in nets. 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, etc. 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 among different metal layers [30]: 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

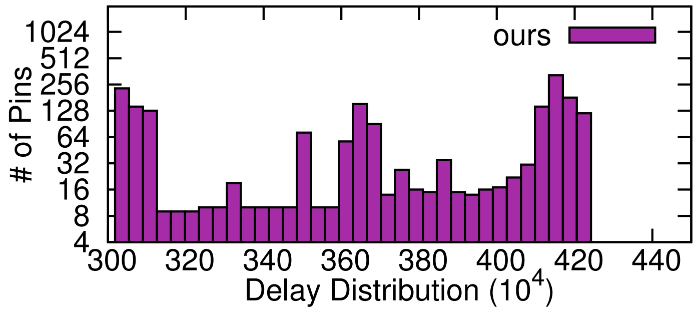
This chapter is based on the journal: Derong Liu, Bei Yu, Salim Chowdhury, and David Z. Pan. "Incremental layer assignment for timing optimization." ACM Transactions on Design Automation of Electronic Systems (2017). I am the main contributor in charge of problem formulation, algorithm development and experimental validations.

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 at minimization of via count, antenna effect avoidance, and timing optimization, etc [7, 20, 44, 45, 54, 76, 91]. 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 [44]. [20] also applies 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, [54] adopts 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 min-cost max-flow model [45]. [7] focuses 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, [91] proposes an incremental layer assignment integrated with timing optimization engine. The



(a)



(b)

Figure 3.1: Pin delay distribution of critical nets for benchmark adaptec1, where 0.5% of the nets are released as critical nets. (a) Results from TILA [91]; (b) Results from our incremental layer assignment framework.

proposed framework, TILA, is able to provide a global view of minimizing the total net delay for the selected nets. As an extension, [50] additionally reduces the slew violations with a control of via overheads. Also, [55] guides the global optimization of timing critical paths by decoupling the layer assignment from timing analysis.

Although TILA [91] can achieve the most state-of-the-art layer assignment results targeting at 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 [71], which is essentially a non-linear 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, critical path timing in each net is more focused, instead of the total sum of net delays.

This chapter proposes a novel incremental layer assignment framework targeting at timing optimization for critical timing paths in nets, where our layer assignment tool is able to achieve better timing optimization. Fig. 3.1 compares the layer assignment results between TILA and our work. To have a clear view of the maximum delay distribution, we start from 3.0×10^6. Fig. 3.1(a) gives the results from TILA, where many pins have delay over 4.2×10^6. On the other hand, from Fig. 3.1(b) we can see that our framework can reduce the maximum delay since the worst pin has the delay around 4.2×10^6. The results have been reported in [49], and 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

speed-up with a post mapping methodology to guarantee integer solutions.

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

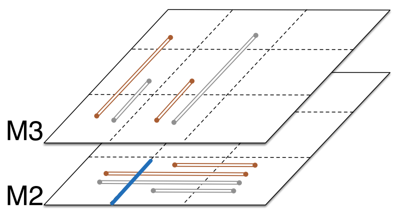
The remainder of this chapter is organized as follows. Section 3.2 provides some preliminaries and the problem formulation. Section 3.3 first presents the mathematical formulation to optimize critical path timing. Then a set of novel techniques are discussed to further achieve a better trade-off between solution quality and runtime. The experimental results are reported in Section 3.4 and the summary is given in Section 3.5.

3.2 Preliminaries

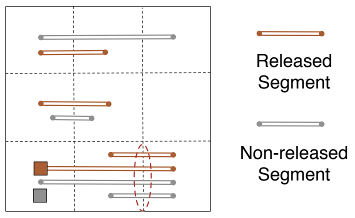
In this section, we give a brief introduction to the graph model and the timing model in our work. Then we propose the incremental layer assignment targeting at critical path timing optimization.

3.2.1 Graph Model

For the graph model adopted in this layer assignment, please refer to the details in Section 2.2.1. In general, each layer supports uni-directional wires and is divided into a set of rectangular tiles, represented by the vertices in the grid model. The edges connecting vertices are divided into two sets: edges in the x/y-direction for wires and edges in the z-direction for vias. For



(a)



(b)

Figure 3.2: Illustration of capacity model. (a) Edge capacity model; (b) Via capacity model.

x/y-direction edges, each of them has a specified routing capacity on different layers, i.e. $cap_e(l)$ 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 $cap_e(l)$. Fig. 3.2(a) provides a detailed illustration of edge capacity model, where the number of wires passing on Metal 2, i.e. M2, should not exceed 4. Notably, for an incremental layer assignment tool, considering that the non-released 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 Fig. 3.2(a), when there is a non-released segment routed on the edge marked as blue on M2, the current edge capacity, $cap_e(l)$, should be set to 3 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, same as [28]. To make it explicit, an illustration of via capacity model is shown in Fig. 3.2(b). Due to the existing non-released nets, we should also take care of those non-released 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 Fig. 3.2(b), the gray square represents a via from a non-released 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 [28],

$$cap_g(l) = \lfloor \frac{(w_w + w_s) \cdot Tile_w \cdot (rcap_{e_0}(l) + rcap_{e_1}(l))}{2 \cdot (v_w + v_s)^2} \rfloor - n'_v, \tag{3.1}$$

where $w_w, w_s, v_w, v_s, Tile_w$ represent wire width, wire spacing, via width, via spacing and tile width, respectively. And n'_v denotes the number of vias from non-released nets. For vias between two layers, each layer have two edges connecting with grid g, i.e. 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 $cap_e(l)$ above which does not consider released segments on edge e, here $rcap_e(l)$ provides the exact residual edge capacity including both released and non-released segments. Therefore, in Eq. (3.1), we can see that the allowable number of vias 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 Fig. 3.2(b), for the circled edge, its $cap_e(2)$ is set to 2 but $rcap_e(2)$ is set to 0 since 2 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, i.e. no residual space.

3.2.2 Timing Model

To calculate the timing cost of each net, we adopt *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. The calculation of segment delay has been introduced in Section 2.2.2. The timing cost of segment depends on its downstream capacitance $C_d(i)$ and the corresponding resistance/capacitance values of its assigned layer.

Then via timing cost is calculated as in Eq. (3.2), which is determined by via resistance and the downstream capacitance of its connected segments [91].

$$t_v(i, j, p, q) = \sum_{l=j}^{q-1} R_v(l) \cdot C_d(V(s_i, s_p)), \tag{3.2}$$

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, i.e. $C_d(V(s_i, s_p))$, 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. And the via delay from two adjacent layers can be calculated based on Eq. (2.2) in Section 2.2.2. Therefore, the integration of via delay is also able to benefit via consumptions.

3.2.3 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 2 (CPLA) *Given a 3-D grid graph, edge and layer information, initial routing and layer assignment, and a set of critical nets, layer assignment reassigns layers among critical and non-critical nets sharing metal resources onto layers in order to minimize their critical path timing while satisfying the edge capacity constraints.*

3.3 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 a 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.

3.3.1 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 as “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 as “non-critical nets”. To make it explicit, both critical and non-critical nets will be reassigned in our framework. Each net is composed of a sequence of segments which 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: Firstly, 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 accounts, 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. Secondly, the same ratio of non-critical nets should also be selected to release high layer resources for those critical nets. Instead of optimizing all the critical and non-critical nets simultaneously in TILA, CPLA places more emphasis on those critical nets which will be assigned at first. After the assignment of critical nets, a greedy method is adopted to assign these non-critical nets in a one-by-one way. Since our optimal target is the maximum path timing of those critical nets, the assignment details of non-critical nets will not be covered but we follow a dynamic programming method in order to control the via counts in [44]. Through optimizing critical and non-critical 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 non-critical nets efficiently, we should search for those nets with the best timing which also share the same edges with the critical ones as much as possible. And the assigned layers of the non-critical nets should be higher than critical nets' for resource releasing. In summary, the selection procedure of non-critical nets is similar as in [91], but with one main difference: here we select the critical/non-critical 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 non-critical nets are selected for reassignment. More details of our proposed ILP formulation are given as follows, while non-critical nets are not included in this formulation. For convenience, notations used are listed in Table 3.1.

Thus, we can obtain the integer linear programming (ILP) formulation as shown in formula (3.3). 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, constraint (3.3b) guarantees that one segment can be assigned on one and only one layer. Constraint (3.3c) sets the routing wire limit for those released segments of edge e on layer j, i.e. $cap_e(j)$. Notably, $cap_e(l)$ not only depends on its initial track number but also those non-released segments passing through e on layer l. As shown in Fig. 3.2(a), when the blue edge on M2 has 4 available tracks initially but one of them has already been occupied by a non-released net, the exact allowable routing capacity, i.e. $cap_e(2)$, should be set to 3. This means that at most 3 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 non-released nets. Thus, for an incremental assignment problem, the edge capacity constraint is more stringent than the initial problem.

Considering the possible existing edge overflows from the input, we

Table 3.1: 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

extend this constraint to comply with both legal and illegal solutions. For legal solutions free of overflows, it is clear to keep $cap_e(l)$ as above, i.e. the initial number of edge capacity excluding those non-released 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 5 routing wires but its capacity should be 4, then an edge overflow does exist. To deal with that, if there are 2 non-released 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, constraint (3.3d) places the limitation of the via number to

$$\textbf{min} \sum_{i \in S(Nc)} \sum_{j=1}^{L} t_s(i,j) \cdot x_{ij} + \sum_{i,p \in Sx(Nc)} \sum_{j=1}^{L-1} \sum_{q=1}^{L-1} t_v(i,j,p,q) \cdot y_{ijpq}, \tag{3.3a}$$

$$\textbf{s.t.} \sum_{j} x_{ij} = 1, \qquad \forall i \in S(Nc), \tag{3.3b}$$

$$\sum_{i \in S(e)} x_{ij} \le cap_e(j), \qquad \forall e \in E, \tag{3.3c}$$

$$\sum_{(i,p) \in Sx(Nc,g)} y_{ijpq} + n_v(x_{ij} + x_{pq}) \le cap_g(l), \quad \forall l, j < l < q, \quad g \in G, \tag{3.3d}$$

$$x_{ij} \ge y_{ijpq}, \qquad \forall (i,p) \in Sx(Nc), \quad j,q \in L, \tag{3.3e}$$

$$x_{pq} \ge y_{ijpq}, \qquad \forall (i,p) \in Sx(Nc), \quad j,q \in L, \tag{3.3f}$$

$$x_{ij} + x_{pq} \le y_{ijpq} + 1, \qquad \forall (i,p) \in Sx(Nc), \quad j,q \in L, \tag{3.3g}$$

$$y_{ijpq} \text{ is binary}, \qquad \forall (i,p) \in Sx(Nc), \quad j,q \in L, \tag{3.3h}$$

$$x_{ij} \text{ is binary}, \qquad \forall i \in S(Nc), \quad j \in L. \tag{3.3i}$$

pass through each grid g for different layers. Similar as $cap_e(l)$, the calculation of $cap_g(l)$ should take those non-released nets into consideration as well. Still, as shown by the lower left grid in Fig. 3.2(b), we assume that the initial via capacity for this grid from M2 to M3 is 16 where each track is able to locate 4 vias and there are totally 4 tracks. Meanwhile, there is already a non-released via passing through M2 and one track is also occupied by another non-released net. Then we reduce the available via space further by deducting the utilized resources of non-released nets, i.e. 5 vias in total. Notably, a newly-assigned segment will also take another available track for routing, thus resulting in further reduction of 4 vias. The final residual space is for routing vias belonging to those released segments. Therefore, in constraint (3.3d), we should consider not only those existing non-released nets but also the newly-assigned segments. Take Fig. 3.2(b) as an example, at most 7 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 Eq. (3.3a), 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 (3.3e)–(3.3g) y_{ijpq} is the product of x_{ij} and x_{pq} because all x_{ij} and y_{ijpq} are binaries according to constraints (3.3h) and (3.3i).

Nevertheless, there is a potential problem for constraint (3.3d). If via capacity violations already exist in initial layer assignment inputs and cannot be eliminated completely, this constraint may be too stringent that no legal solutions can be obtained. To avoid this condition, we relax this constraint by adding a slack variable V_o, representing the number of maximum allowable violations. Then constraint (3.3d) can be re-written 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 2000 in our implementation. Thus, the ILP formulation can guarantee reasonable solutions with legal edge capacities and controllable via violations. Similar to [91], 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 huge calculation overhead with the considerable runtime. In order to alleviate this overhead, speed-up techniques are introduced in the following sections.

3.3.2 Self-Adaptive Partition Algorithm

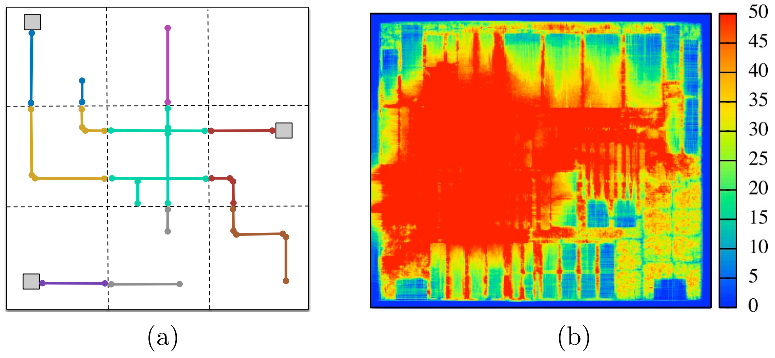


Figure 3.3: Example of grid partition. (a) Nets partition; (b) Routing density for benchmark adaptec1 by NCTU-GR.

For layer assignment work, the routing wires are adjusted in 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 [25], the newly updated assignment results of neighboring partitions benefit each current partition. Fig. 3.3(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, Fig. 3.3(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 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.

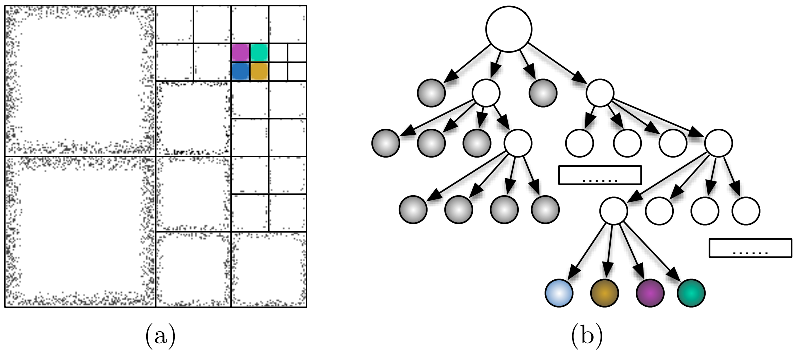


Figure 3.4: Sub-grid partition illustration. (a) Sub-grid partition; (b) Sub-grid corresponding partition tree.

Fig. 3.4(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, Fig. 3.4(b) shows the quadruple tree corresponding to Fig. 3.4(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 Fig. 3.4(a). There are two leaf nodes in the first level representing these two left partitions. In Fig. 3.4(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.

3.3.3 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 with an opportunity for further speed-up. 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 [79], and thus it has been applied in many circuit design problems, such as circuit sizing [80], high-level synthesis [18], power/ground network optimization [24,34], and layout decomposition [40, 92]. To the best of our knowledge, this is the first work to adopt SDP to solve layer assignment problem. We re-write the formulation into the following standard SDP form:

$$\begin{aligned} \textbf{min} \quad & \mathbf{T} \bullet \mathbf{X}, && \text{(3.4a)} \\ \textbf{s.t.} \quad & \mathbf{C} \bullet \mathbf{X} = \mathbf{b}, && \text{(3.4b)} \\ & \mathbf{X} \succeq \mathbf{0}. && \text{(3.4c)} \end{aligned}$$

where

$$\mathbf{T} \bullet \mathbf{X} = \sum_{i \in S(Nc)} \sum_{j \in L} T_{ij} X_{ij}. \tag{3.5}$$

In Eq. (3.4), matrices $\mathbf{T}$ and $\mathbf{X}$ are both $|S \cdot L|$-dimension symmetric matrices, where $|S|$ is the number of segments belonging to the critical nets in each partition and $|L|$ is the number of layers. In Eq. (3.5), $\mathbf{T} \bullet \mathbf{X}$ 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.

Eq. (3.6) shows all coefficients in matrix $\mathbf{T}$, where the items on the diagonal line represent the timing costs, i.e. $t_s(i, j)$, for assigning segment i on layer j. Besides, $t_v(i, j, p, q)$ is the via cost on assigning segments i and p onto 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 Eq. (3.7) 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}.

$$\mathbf{T} = \begin{pmatrix} t_s(i,j) & \dots & t_v(i,j,p,q) \\ \dots & \dots & \dots \\ t_v(i,j,p,q) & \dots & t_s(p,q) \end{pmatrix}, \tag{3.6}$$

$$\mathbf{X} = \begin{pmatrix} x_{ij} & \dots & y_{ijpq} \\ \dots & \dots & \dots \\ y_{ijpq} & \dots & x_{pq} \end{pmatrix}. \tag{3.7}$$

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 constraints (3.3e)–(3.3g), which also apply for continuous solutions. Because constraints (3.3e)–(3.3g) are mainly inequalities, then 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 Eq. (3.3a)).

To guarantee an effective solution, the constraints in ILP formulation (3.3) should also be included in SDP through (3.4b). Constraints (3.3b) and (3.3c) 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 details for the first constraint (3.3b). Evidently, a set of coefficient matrices, $\mathbf{C_b}$s, is required, and the set size is equal to the number of segments in the critical nets belonging to 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 constraint (3.3b), 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 (3.4b) 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 (3.3c), 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 constraint (3.3d), we prefer to move it into the objective matrix by adding the penalty to save the runtime. Then the 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 presented layer assignment problem. Fig. 3.5 shows a part of one net. Due to the space limitation, 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 x-dimension, while layer 2 and 4 for y-dimension. Thus, the matrices $\mathbf{T}$ and $\mathbf{X}$ should be both 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

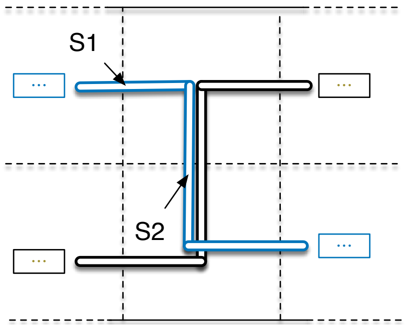


Figure 3.5: An example of layer assignment through SDP.

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

Figure 3.6: **T** matrix and solution **X** matrix of the example.

with x_{ij} and x_{pq} represent the potential via costs from layer j to layer q. Based on Fig. 3.5, s_1 only connects with s_2, so we just need to consider the via costs between s_1 and s_2.

In Fig. 3.6 we list an example of matrix **T**, as well as matrix **X** after solving the SDP. For matrix **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.

3.3.4 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 3 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 3, 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 more competitive for segments to assign. Since edges are divided into x-dimension and y-dimension 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, i.e. $cap_e(j)$. This means that the number of those released segments to assign should not exceed this constraint. Here we select 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.

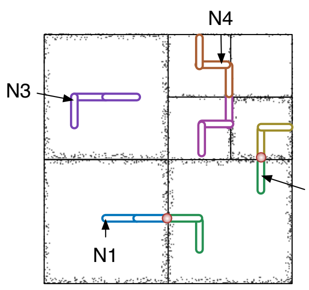
3.3.5 Concurrent Matching Algorithm

The mapping algorithm proposed in Section 3.3.4 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, i.e. s_1 and s_2, 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 still lacks a global view of optimizing all nets to some extents. To handle this conflicting assignment on limited resources, in this section we propose a concurrent matching methodology, which adopts new rounding strategies and targets more concrete solutions.

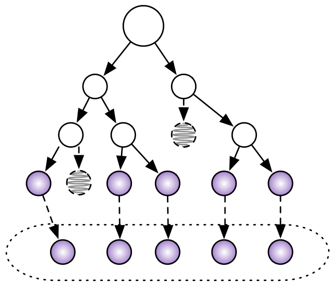
From the sequential algorithm, we obtain only one exact assignment for each edge based on non-integer solutions. Nevertheless, when the number of released segments on one edge is very high, keeping on one assignment may lose some potential optimal solutions. In the following, we present how to produce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

As depicted in Section 3.3.2, 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. Fig. 3.7(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, as shown by 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 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 solutions mapping. The first step, how to generate a set of candidate solutions according to the floating-point solutions, is deserved to explore. A procedure of candidate generation for segments in net N_3 is shown in Fig. 3.7(b), where we adopt a top-down method to traverse all the possible assignments of net N_3. Since there are totally three segments for N_3, three levels are essential to seek 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 value 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 Fig. 3.7(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 sat-



(a)



(b)

Figure 3.7: Example of solution candidate generation. (a) Nets partition; (b) Solution candidate generation for N_3.

isfy the required capacity constraint. Through these two bounding methods, the solution space is controlled efficiently, and possible long runtime overhead can be prevented. In Fig. 3.7(b), the bottom circle indicates a set of candidates for the division where net N_3 is. For the divisions with a significant number of solutions, we sort the solutions based on their internal via violation costs, and the best 20 solutions with the least via violations are taken as final candidates for such a division.

With these generated solution candidates, we formulate our solution selection problem as an integer linear programming (ILP) as in Eq. (3.8). The notations are listed with details in Table 3.2. Explicitly, ILP formula (3.8) targets at optimizing both division's internal via violation costs and its external costs with neighboring divisions. Since timing has been devoted to much attention throughout candidate generation, only vias and via violations are counted as elements of costs in Eq. (3.8a). For a solution n of division m, its corresponding cost, $c_d(m, n)$, is the sum of its internal vias and violations, both

Table 3.2: 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

of them multiplied with weights; while considering vias on the boundary between two divisions m and u, as shown in red points in Fig. 3.7(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 (3.8b) guarantees that only one solution can be selected for each division, D_m. Meanwhile, constraints (3.8c)–(3.8e) limit that b_{mnuv} is the product of a_{mn} and a_{uv}, based on the condition that a_{mn} is binary from constraint (3.8g). Its form is very similar to Eq. (3.3), due to their essence of solving the same assignment problem, and thus we save the very detailed description of this formulation.

$$\begin{aligned}\mathbf{min} \quad & \sum_{m \in D(Nc)} \sum_{n \in Sol(m)} c_d(m,n) \cdot a_{mn} + \\ & \sum_{m,u \in Dx(Nc)} \sum_{n \in Sol(m)} \sum_{v \in Sol(u)} c_{dx}(m,n,u,v) \cdot b_{mnuv}, \qquad (3.8a)\end{aligned}$$

$$\begin{aligned}\textbf{s.t.} \quad & \sum_n a_{mn} = 1, && \forall m \in D(Nc),\ n \in Sol(m), && (3.8b)\\ & a_{mn} \geq b_{mnuv}, && \forall (m,u) \in Dx(Nc),\ n,v \in Sol(m), Sol(u), && (3.8c)\\ & a_{uv} \geq b_{mnuv}, && \forall (m,u) \in Dx(Nc),\ n,v \in Sol(m), Sol(u), && (3.8d)\\ & a_{mn} + a_{uv} \leq b_{mnuv} + 1, && \forall (m,u) \in Dx(Nc),\ n,v \in Sol(m), Sol(u), && (3.8e)\\ & b_{mnuv} \text{ is binary}, && \forall (m,u) \in Dx(Nc),\ n,v \in Sol(m), Sol(u), && (3.8f)\\ & a_{mn} \text{ is binary}, && \forall m \in D(Nc),\ n \in Sol(m). && (3.8g)\end{aligned}$$

To make this whole process more clear, we provide an overall algorithm flow including the proposed algorithm flow in Fig. 3.8. After the selection of critical nets and solving SDP, as shown in the left block, the iterative flow assigns segments through the sequential method to ensure the convergence of SDP solutions. Consider that if we integrate this concurrent algorithm into the same flow, the runtime overhead may be non-negligible 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

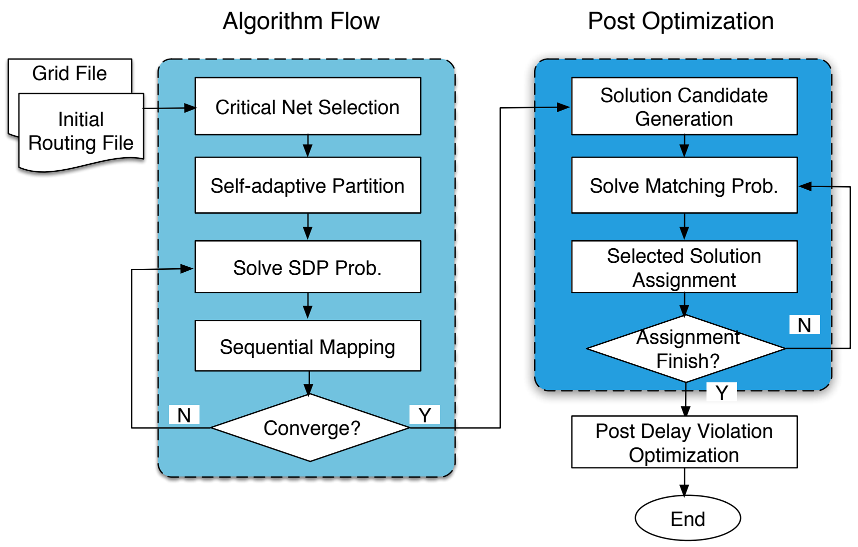


Figure 3.8: Algorithm flow including matching algorithm.

to take more possibilities into accounts. Through the presented generation methodology, a few candidate solutions are generated for each partition based on their obtained SDP solutions. Due to the massive number of divisions, we prefer to relax the Eq. (3.8) 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 non-integer 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 solution, when the number of residual divisions is small enough, i.e. smaller than 2000, 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.

3.3.6 Post Delay Optimization

The algorithm flow aforementioned gives a global view of timing optimization for critical path timing in the nets, where these critical nets are selected as those with 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 4.

The outline of the post delay violation algorithm is listed in Algorithm 4, while the notations are also listed in Table 3.2. 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 at reducing potential delay violations for a specified timing constraint. Thus, without over-utilizing high layer resources for those 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 the nets with violations, which complies with the ILP formulation. In our future work, we will consider timing

Algorithm 4 Post Delay Violation Algorithm

```
Input: A set of critical paths $P_c$;
1: Sort $p$ with decreasing $t(p_c)$;
2: for each $p_c \in P_c$ do
3:     for each $s_c \in p_c$ do
4:         for $j = L_m$; $j \geq 1$; $j = j - 2$ do
5:             if $j \leq l(s_c)$ 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 $t(p_c) \leq T_0$ then
18:            Break;
19:        end if
20:    end for
21: end for
```

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 the decreasing order of their critical path timing (line 1). By starting from the path which 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, same as Algorithm 3, for a switching segment, s_s, which exists on a non-critical 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 5–7, 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 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 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 the next critical path. Through this improvement, delay violations can be reduced explicitly with certain timing constraints.

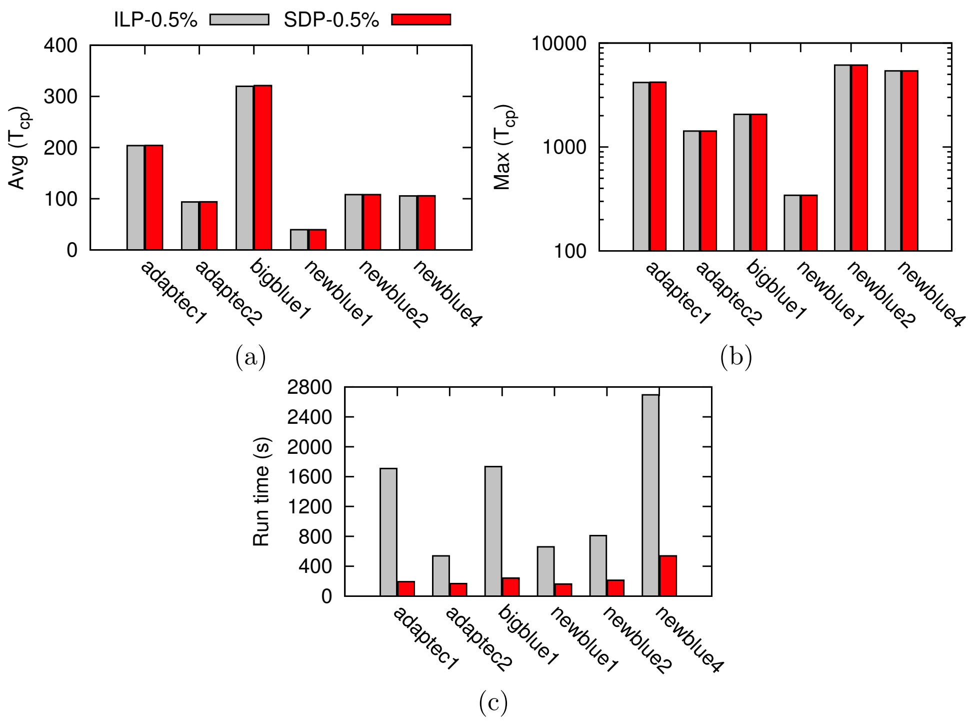


Figure 3.9: Comparison between ILP and SDP on some small test cases: (a) on average delay for all critical paths; (b) on maximum delay for all critical paths; (c) on runtime.

3.4 Experimental Results

3.4.1 Timing Results

The proposed layer assignment framework is implemented in C++, and tested on a 32-core Linux machine with 2.9 GHz Intel® Core and 192 GB memory. We select GUROBI [26] as the ILP solver, and CSDP [10] as the SDP solver. Besides, we utilize OpenMP [11] for parallel computing, and set the thread number to 16. As that in [91], we test our framework on ISPD 2008 global routing benchmarks [65]. It should be noted in our experiments, 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 3.3.1) with the SDP-based methodology (see Section 3.3.3 and Section 3.3.4). Since ILP formulation may suffer from runtime overhead problem, i.e., it cannot finish in two hours for some large test cases, we select some small test cases for the comparison as shown in Fig. 3.9. Note that the partitioning technique is applied to both methods. We can see from Fig. 3.9(a) and Fig. 3.9(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 speed-up (see Fig. 3.9(c)).

In the second experiment, we further evaluate our SDP-based method by comparing it with TILA [91]. To make a fair comparison, we release the same set of nets for both TILA and our SDP. Table 3.3 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 3.3 we can see that compared 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

Table 3.3: Performance comparison with TILA on ISPD 2008 benchmarks.

bench	TILA-0.5% [91]					SDP-0.5%				
	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	4378.42	49205	19.31	132.9	204.88	4205.71	50947	19.26	112.5
`adaptec2`	97.94	1435.79	38173	19.25	133.8	93.88	1421.68	38480	19.32	91.2
`adaptec3`	220.00	4613.89	90961	36.74	322.5	209.41	4583.29	92299	36.76	569.0
`adaptec4`	121.67	5616.23	72695	32.22	272.4	117.43	5590.84	73185	32.44	494.3
`adaptec5`	249.51	5406.11	81151	55.21	444.8	216.15	5311.75	84537	55.26	472.0
`bigblue1`	402.81	2673.18	44399	21.69	174.7	322.41	2065.42	46256	21.56	142.1
`bigblue2`	100.94	10821.67	114343	43.38	188.7	95.58	10728.23	115240	43.49	264.9
`bigblue3`	27.38	789.61	65718	52.62	333.5	21.53	373.80	66795	52.92	547.7
`bigblue4`	37.98	3779.11	95348	109.94	747.1	33.56	3750.95	97148	110.37	804.3
`newblue1`	43.11	344.32	57063	22.34	106.9	39.52	343.09	57744	22.44	98.7
`newblue2`	110.76	6171.37	35994	28.97	151.4	107.85	6130.09	35566	29.25	146.4
`newblue4`	111.53	5660.31	84684	47.57	305.9	105.53	5395.42	85159	47.73	365.2
`newblue5`	170.45	2789.52	152770	86.65	605.1	151.41	2771.55	157944	87.00	1564.4
`newblue6`	144.42	2373.86	94489	78.47	683.4	124.75	2298.74	97859	78.53	562.2
`newblue7`	30.03	1301.30	143087	163.81	1161.2	25.33	1254.22	144580	164.28	1555.7
average	139.81	3896.98	81339	54.54	384.3	124.61	3748.32	82916	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

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, due to the nature that the SDP problem is more complicated than the min-cost flow problem. However, since we adopt the adaptive partitioning in the SDP-based method (see Section 3.3.2), this method can still achieve reasonable runtime. During partitioning, we set its allowed number of segments in each partition as 10 for the further self-adaptive partitioning methodology.

In the third experiment, we demonstrate the effectiveness of our self-adaptive partitioning methodology for SDP, as shown in Fig. 3.10. We try dif-

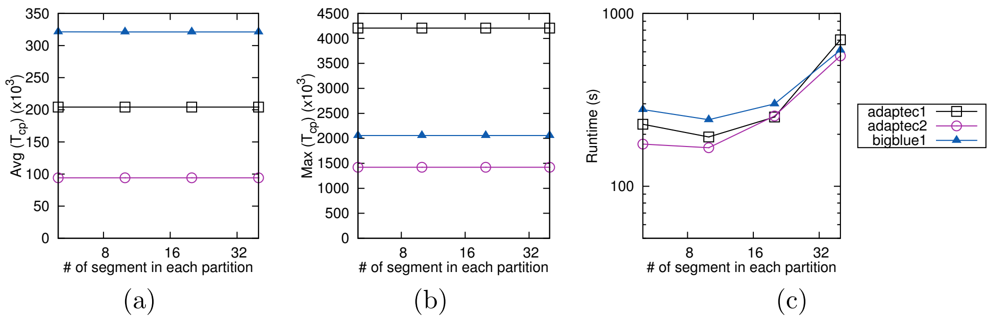


Figure 3.10: Partition size impact on three small cases. (a) The impact on Avg(T_{cp}); (b) The impact on Max(T_{cp}); (b) The impact on runtime.

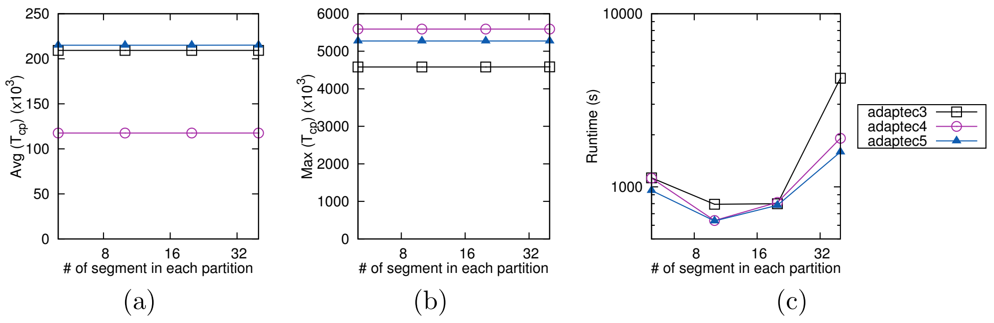


Figure 3.11: Partition size impact on three large cases. (a) The impact on Avg(T_{cp}); (b) The impact on Max(T_{cp}); (b) The impact on runtime.

ferent partition granularities (from 5 to 40) for three small test cases, where the maximum number of segments in each partition is limited. From Fig. 3.10(a) and Fig. 3.10(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, Fig. 3.10(c) shows that the runtime increases drastically with the partition granularity. Notably, without the self-adaptive partitioning methodology, the number of critical segments to deal with is so high that it takes around 1000 seconds to run even a small benchmark by releasing 0.5%. Therefore, we can see that the self-adaptive partitioning methodology benefits the runtime for SDP significantly. Meanwhile, we can observe that when the constraint is set to 10, 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 Fig. 3.11. 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 similar runtime for benchmark “adaptec3”, and 20-segment granularity can even spend less runtime than 5-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 trade-off 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 number of tasks. 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.

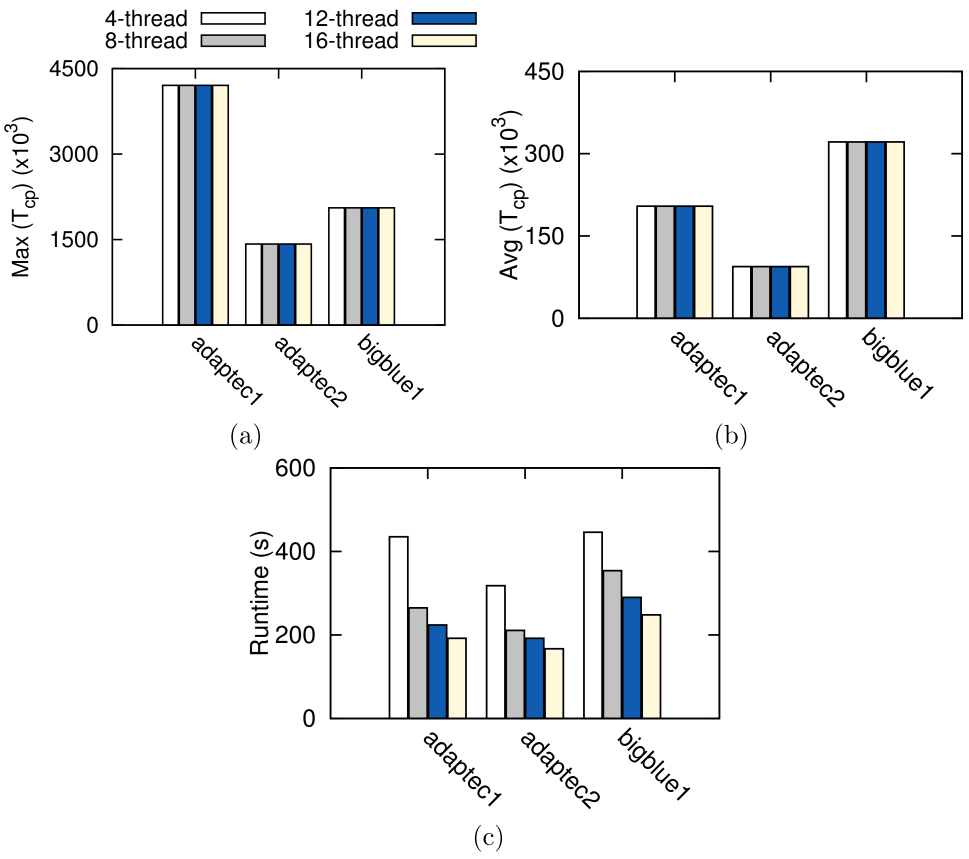


Figure 3.12: Performance evaluation based on the number of threads on some small test cases: (a) Maximum delay for critical paths; (b) Average delay for critical paths; (c) Runtime.

In the fourth experiment as given in Fig. 3.12, we evaluate the impact of thread number with the same partition granularity on three small benchmarks. As limited by machine resources, the number of threads can be lower than 16 and higher than 4, so we select four numbers, i.e. 4, 8, 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

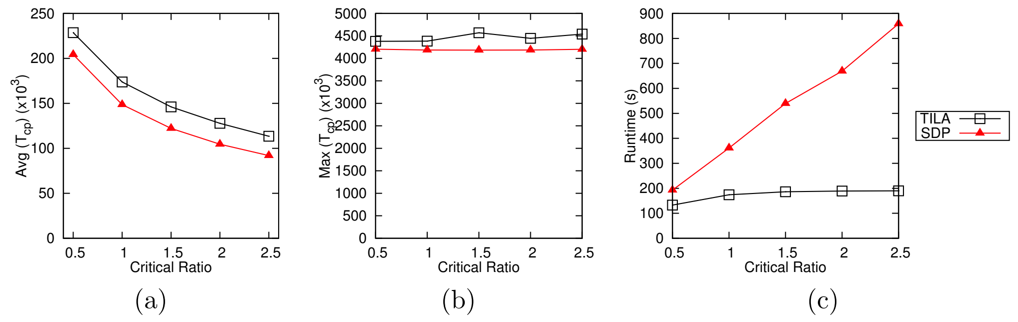


Figure 3.13: Critical ratio impact on benchmark `adaptec1`. (a) The impact on Avg(T_{cp}); (b) The impact on Max(T_{cp}); (b) The impact on runtime.

shown in Fig. 3.10 and Fig. 3.11. 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 speed-ups; but when the number reaches a certain threshold, the speed-up 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 most speed-ups.

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 3.4. From Table 3.4, 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

Table 3.4: Performance comparison with/without post opt.

bench	WO Post		W Post	
	Max(T_{cp}) (10^3)	# of Vio	Max(T_{cp}) (10^3)	# of Vio
`adaptec1`	4205.71	6215	4205.71	5474
`adaptec2`	1421.68	5148	1421.68	3820
`adaptec3`	4583.44	9812	4583.40	8367
`adaptec4`	5591.10	11091	5591.10	8429
`adaptec5`	5311.75	21273	5273.78	17583
`bigblue1`	2065.42	12871	2056.57	11329
`bigblue2`	10728.70	18211	10725.35	16004
`bigblue3`	373.80	5122	304.20	1149
`bigblue4`	3751.01	10451	3751.06	2294
`newblue1`	343.09	8282	343.09	7044
`newblue2`	6130.09	11065	6130.09	6961
`newblue4`	5395.42	12599	5395.42	9327
`newblue5`	2771.74	40471	2771.74	32196
`newblue6`	2298.74	15288	2298.74	11286
`newblue7`	1254.22	13823	1254.22	4555
average	3748.39	13448	3740.41	9721
ratio	**1.000**	**1.000**	**0.998**	**0.723**

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 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 on the performance of the SDP-based method. Critical ratio is an important parameter to determine how many critical nets are released. In Table 3.3, 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 Fig. 3.13(a) and Fig. 3.13(b), we see that the average timing decreases slightly with the increase of critical ratio for both SDP and TILA. However, for the comparison of maximum timing, 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 Fig. 3.13(c), we observe that for the SDP-based method the runtime increases in proportion to the critical ratio. This illustrates that our method has a well-controlled scalability.

3.4.2 Timing Violation Results

The third last experiment manifests the effect of our post stage to reduce timing violations. To compare with CPLA results in [48], 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 criterion based on their initial timing conditions. Therefore, we set the delay constraint to 80% of the minimum timing of those critical nets in Table 3.3. Table 3.5 lists the compared delay violations, also

Table 3.5: Delay violation comparison on ISPD 2008 benchmarks.

bench	CPLA in [48]						CPLA New					
	# 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

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 vias costs and violations are also well controlled. The only penalty we pay is the 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 the 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 in Fig. 3.14(a), around 2% of via violations can be reduced; meanwhile, the number of via violations decreases by 0.34% in Fig. 3.14(b). Therefore, under a reasonable runtime overhead, the concurrent matching strategy is deserved.

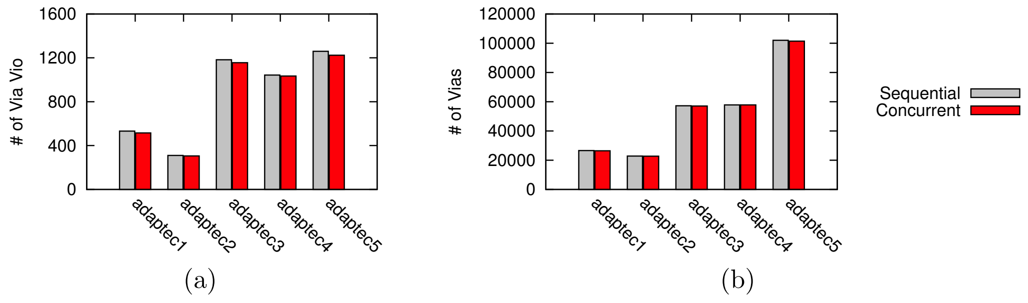


Figure 3.14: Comparison between sequential mapping and concurrent matching: (a) via violations; (b) number of vias.

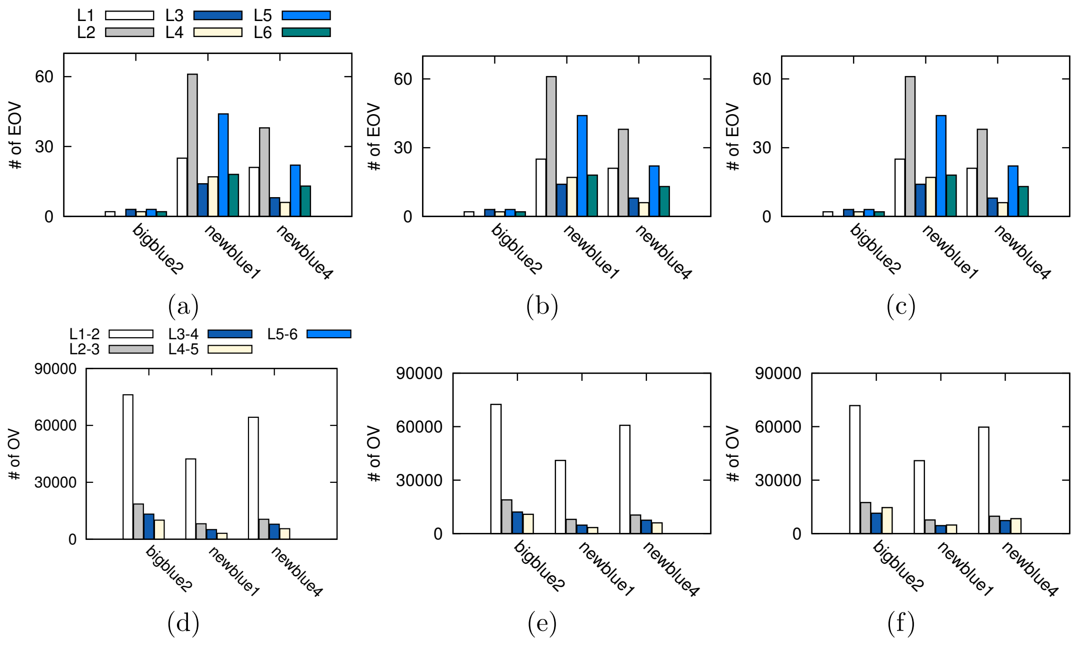


Figure 3.15: Violation comparison of NVM, TILA and CPLA: (a) Edge overflows in NVM; (b) Edge overflows in TILA; (c) Edge overflows in CPLA; (d) Via overflows in NVM; (e) Via overflows in TILA; (f) Via overflows in CPLA.

Besides, to provide an explicit view of edge/via overflow distribution among layers, we measure the number of edge overflows on each layer and via violations between every two adjacent layers for NVM [54], 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 Fig. 3.15(a), Fig. 3.15(b) and Fig. 3.15(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 Eq. (3.2), 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 the 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 Fig. 3.15(f) shows slightly higher than that in Fig. 3.15(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 via violations exist on the lowest and highest layer.

3.5 Summary

This chapter targets at optimizing critical path timing during the layer assignment stage. First, we propose the ILP formulation for the problem, and then present the self-adaptive partition algorithm to benefit the runtime. Based on this partition algorithm, the SDP-based method is developed and applied to each division. Additionally, an iterative LP framework is integrated as the post stage with an 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, and little performance degradation is observed from multiple threads but with much speed up. With the post delay optimization, timing violations can also be reduced efficiently.

Chapter 4

Synergistic Topology Generation and Route Synthesis for Signal Groups

4.1 Introduction

The previous chapters have introduced two layer assignment approaches, which aim to adjust the assigned layers for timing optimization. This chapter provides an extensive view to develop competitive topology generation and route synthesis for on-chip interconnections. In current industrial designs, data and control signals loading messages from various sources can be bound as signal groups, as shown in Fig. 4.1. Observe that there are three signal groups marked with different colors. The signal bits in one group may have different numbers of pins, resulting in different routing styles. In this example, one style is signified with a pair of solid lines and a dashed line in the middle. The solid lines represent two signal bits on the border, as pointed in Fig. 4.1, while the dashed lines represent multiple bits inside. For signal bits with different pin locations in one group, they have to be routed in a

This chapter is based on the journal: Derong Liu, Bei Yu, Vinicius Livramento, Salim Chowdhury, Duo Ding, Huy Vo, Akshay Sharma, and David Z. Pan. "Synergistic topology generation and route synthesis for on-chip performance-critical signal groups." IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (2018). I am the main contributor in charge of problem formulation, algorithm development and experimental validations.

regular manner. That is to say, common topologies are preferred to be shared among all the bits for design regularity, which is an extension of classic bus routing [42, 61, 70]. Meanwhile, with more metal layers integrated, it faces more challenges to control the routing congestion among multiple layers. For the performance-critical signal bits, the routability and wire-length should also be optimized to avoid functional inaccuracy and timing issues. Therefore, an advanced synergistic router should be able to not only control routability and wire-length but also guide each bit routing intelligently for design regularity.

To realize these requirements, we propose an automatic topology generation and synthesis engine which is able to guide the routing of signal groups with a global view. Besides the improvement of routability and wire-lengths for the signal bits, we should also pay attention to the specific constraints brought by signal groups, where the bits in one group are encouraged to be routed in parallel tracks and share common topologies/layers for regularity. Meanwhile, instead of a bit-by-bit routing, signal bits can be clustered based on their possible route styles, as seen in Fig. 4.1, where two styles in $Group1$ are circled to be treated as an individual object. Then the problem size can be reduced by condensing several bits into an object, but with the resulting parallel routes, capacity constraints become more stringent. During the whole procedure, all these constraints should be taken into accounts carefully.

There are a few previous works focusing on bus architecture synthesis for on-chip designs. Some bus-oriented work incorporates with floorplanning to satisfy the timing constraints [66], minimize total bus area [86], or improve

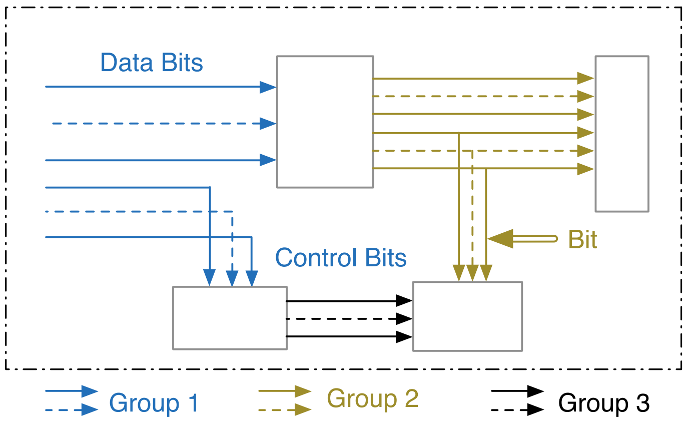


Figure 4.1: Example of on-chip signal groups.

dead space [58]. For timing targeted analysis, an automated bus synthesis framework, FABSYN, incorporates the floorplanning with wire delay estimation engine to detect the potential timing violations [67]. Considering the impact of vias on lithography, a revisiting methodology is proposed to minimize the routing vias while controlling the loss of chip area and wire-length [27]. Furthermore, an OPC-friendly bus floorplanning algorithm allocates the bus positions with the consideration of the impact of Off-Axis Illumination (OAI) on pitches [85]. Especially, multi-bend shapes are considered in [58] for providing more topology candidates through simulated annealing. Additionally, an effective algorithm minimizes the deviation for large-scale buses while improving the dead spaces and wire-length [82]. And a bus thermal analyzer models the potential hot spots on chips [38]. There is also some literature about escape routing on printed circuit board (PCB) design: such as pin ordering and untangling [88], layer resource minimization [87], and an automatic planning flow in [81] including bus decomposition, escape routing, layer assignment, and global routing. Compared with the previous works, our proposed routing synthesis tool provides a more extensive view to deal with bundled signal groups with more possibilities.

Very few of the previous routing works target at synergistic topology generation and routing synthesis of signal groups with multi-pin connections. For current industrial designs, regular topologies with parallel routes are highly preferred to reduce inter-bit variability spread on silicon. Therefore, an efficient topology generator should be able to facilitate the routes of signal bits directing to different cells with low twisting or distorted connections. Besides, compared to two-pin buses, signal groups contain the bits with varying numbers of pins according to their specified logic connections. This also increases the problem complexity by providing more routing possibilities and congestion challenges. Additionally, considering the requirement of source-to-sink distance control, appropriate twisting routes are required to complement the deviation among multiple bits in groups. Therefore, an intelligent framework is essential to guide the routing with the respect of regularity and wire-length efficiently.

In this chapter, we propose an automatic topology generator and routing synthesis flow for on-chip performance-critical signal groups. The results have been reported in [51], and the contributions are highlighted as follows.

- An automatic framework directs topology and routing synthesis of bundled groups with multi-pin connections.

- An identification stage partitions signal groups into a set of objects where each bit has an equivalent topology.
- A mathematical formulation improves routability and wire-length while handling the topology similarity.
- A primal-dual flow benefits the runtime while keeping very comparable quality.
- A bottom-up clustering strategy integrates with layer prediction to enhance the routability of signal groups.
- A refinement stage allows appropriate twisting routes to reduce the source-to-sink distance deviation.

The remainder of this chapter is organized as follows. Section 4.2 presents the overview of our framework and adopted models. Section 4.3 describes our synergistic topology generation procedure, presents a mathematical formulation to optimize wire-length and routability while controlling regularity, and a prime-dual flow benefits the runtime. Section 4.4 provides a post optimization stage to further enhance the signal routability and match the source-to-sink distances among different bits. Section 4.5 reports the experimental results, and followed by conclusion in Section 4.6.

4.2 Preliminaries

In this section, we provide the overview of our proposed framework, and illustrate the adopted model and methodology, based on which a problem formulation is given.

4.2.1 Streak Flow

To provide an explicit view of Streak framework, the overall flow is illustrated in Fig. 4.2. Initially, the information of specified track allocation and pin locations from bits bundled in signal groups is provided. To make it explicit, the definition of signal groups is given as follows, and signal groups are pre-defined and provided by users:

Definition 1 (Signal Group) *The performance-critical signal bits whose pins are located in adjacent physical locations and required to share common topologies are defined as a signal group.*

Considering that the bits in a group may require various routing types, as shown in Fig. 4.1, we identify the possible routing types of each bit based on its pin locations. Those bits are combined as one routing object and able to obtain equivalent topologies. Since our framework targets at multi-layer structure, 3-D topology candidates are required for the objects on different layers. To achieve this, we construct a set of 2-D backbones for each object and derive equivalent topologies for each bit in an object. In our flow, a backbone contains a complete routing solution of all the bits in reference to this

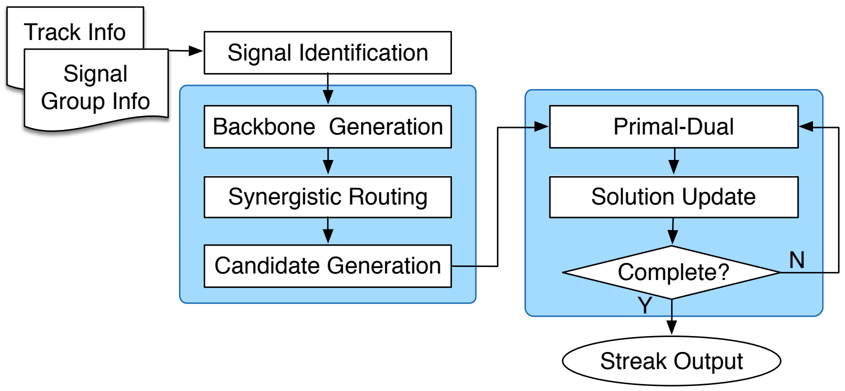


Figure 4.2: Overall Streak flow.

backbone. Then the acquired topology candidates are developed to different layers for a 2-D solution, all of which are considered as candidates for selection. After handling the equivalence of each object, we further quantify the dissimilarity among objects in a group through regularity ratio. Based on these operations, a primal-dual flow solves all the objects efficiently. During the primal-dual flow, we search for the most appropriate solution for each signal routing object in a progressive manner. The details of each step in the flow will be given in Section 4.3. The following sections describe the routing model adopted through our framework to handle the topology generation and route synthesis for synergies.

4.2.2 Proposed Signal Model

Similar to global routing, a signal route can also be modeled on a 3-D global grid model. In real industrial designs, 3-D routing is preferred to avoid the sub-optimality of a post layer assignment step. Similarly, each layer is

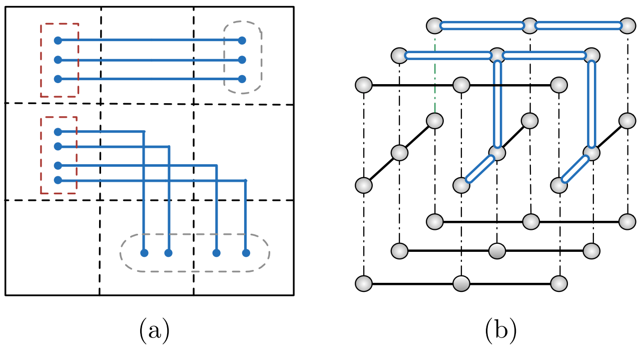


Figure 4.3: Illustration of signal routing model: (a) Example of 2-D routing; (b) Example of 3-D routing.

also divided into a set of rectangular routing cells in a 2-D manner, i.e. **G-Cell**, shown as a vertex in Fig. 4.3(b). Additionally, the edges connecting vertices in 2-D planes are for routing wires, whose capacity constraints have to be satisfied. This means that the number of passing bits cannot exceed the maximum capacity for each edge. Different from traditional routing, signal bits prefer to be routed in parallel tracks and share common topologies as much as possible for regularity. For a signal group, several bits may occupy the same edge simultaneously, which aggravates the routing congestion. Therefore, edge capacity constraint becomes more challenging through guiding the overall route of all signal bits.

Based on the 3-D grid model, efficient routes can be generated considering the specified requirements of signal groups. By modeling $Group1$ from Fig. 4.1 on a 2-D grid, as shown in Fig. 4.3(a), this group is to be divided into two routing styles based on their pins' locations as circled. Each style corresponds to an individual object consisting of several bits, and the topologies of these two objects are encouraged to be shared as much as possible. Therefore, it turns out that they are routed in horizontal tracks from the drivers, and their corresponding 3-D solutions are provided in Fig. 4.3(b), where the horizontal trunks are assigned on the same metal layer.

4.2.3 Proposed Bit Model

As shown in Fig. 4.1, one signal group contains a specified number of bits, which may have various numbers of pins located in different directions. For the bits belonging to one group, their routes should be coordinated and adhere to some constraints: topology variance should be well controlled by matching the connection with the mapped pins located at the similar direction in the bits; for a pair of mapped pins, their source-to-sink distances should be within certain bounds to reduce the deviations. As shown in Fig. 4.4(a), three bits are listed with the mapped pins as clustered together, where the red dashed squares signify the drivers of all the bits. Observe that the distances between the driver and each mapped pin are all the same though these bits possess different numbers of pins. Nevertheless, not all the bits in one signal group are able to achieve the equivalent distance ideally. An example is provided in Fig. 4.4(b), from which a much shorter distance exists for one sink in the leftmost bit compared to the other bits. This will induce the source-to-sink distance deviation for this mapped pin, and further result in non-negligible diverse arrival times for connecting modules. To avoid the possible resulting malfunction for these modules, it is essential to control the distance deviation

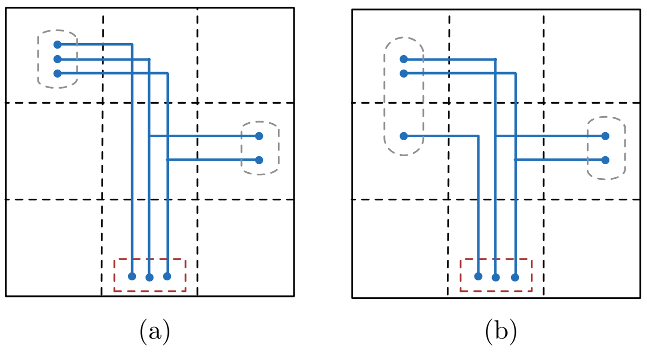


Figure 4.4: Illustration of source-to-sink distance for signal bits: (a) Example of equivalent distance for all bits; (b) Example of inter-bit distance deviation.

in an acceptable range. Therefore, a deviation threshold is introduced so that the deviation should be under this constraint.

4.2.4 Proposed Similarity Vector Model

Based on the examples in Fig. 4.3 and Fig. 4.4, it is imperative to present a model which distinguishes the bits in a bundled group according to their different pin connections. That is to say, all the bits in a distinguished object are to acquire equivalent topologies. FLUTE [16] provides an elegant definition of equivalent topology through vertical sequences, where the same sequence is guaranteed to produce an equivalent topology. By extending this, we develop a similarity vector for each pin, $SV(p_m)$, to capture its relative location in its bit. Furthermore, $SV(p_m)$ is also utilized to find the corresponding pin in another bit from one signal group. Based on the corresponding pins from other bits, their routes can be coordinated in a synergistic manner through appropriate mapping and calibration.

Since the corresponding pins of different bits can be located in various **G-Cell**s, we prefer to use the relative direction rather than distance to describe each pin's location. As shown in Fig. 4.5(a), the SV for pin p_m in it's net is decided through a quadrant-based model, which characterizes the connecting directions in comparison to p_m. It is seen that there are 8 directions in total: each quadrant contributes a direction while both X and Y axes contribute two directions. Then a similarity vector is presented as shown in Equation (4.1),

$$SV(p_m) = \{n_p(+x), n_p(I), n_p(+y), \cdots, n_p(IV)\}, \tag{4.1}$$

which records the number of other pins in this bit, i.e. n_p, from each direction by a counter-clockwise sequence. For the example shown in Fig. 4.5(a), assume that the driver is in the middle and each "X" represents a sink, then SV of this driver is {1,1,1,1,1,1,1,1}. Taking the example in Fig. 4.3(a) as an instance, there are two routing styles which can be distinguished through the vector. For the top style, the SV of the driver is {1,0,0,0,0,0,0,0}, while the sink has the SV as {0,0,0,0,1,0,0,0}. Thus, in each routing style, every pin has the same SV, and these pins belonging to various bits in a group can be mapped mutually. Based on the mapped pins, we are able to provide equivalent topologies for the bits in an object, while the topologies among different objects can also be coordinated to reduce the dissimilarities. Therefore, SV plays an important role in processing topology synergy of the bits in a group.

4.2.5 Problem Formulation

Based on the proposed flow and routing model discussed in the preceding section, we define the synergistic topology generation and route synthesis (Streak) problem as follows:

Problem 3 (Streak) *Given signal bits in bundled groups and layer capacity information, Streak determines the routing topology and layer assignment for each signal bit so that the routability, wire-length and topology regularity can be optimized while the edge capacity constraints are satisfied.*

4.3 Algorithms

In this section, we present the technique details adopted through Streak flow. A pre-processing stage partitions each signal group into a set of routing objects; a set of backbone structures is constructed and equivalent topologies are developed; a mathematical formulation selects the appropriate topology and assigns penalties to control irregular topologies; and a primal-dual algorithm is presented finally for speed-up.

4.3.1 Identification of Signal Isomorphism

Besides covering general bus routing, our framework provides more feasibilities to handle groups of signal bits, which can be loaded from data or control information. A pre-processing stage provides a set of bits clustered in different groups, which can belong to multiple buses and prefer to share common topologies as much as possible. In comparison to general bus plan-

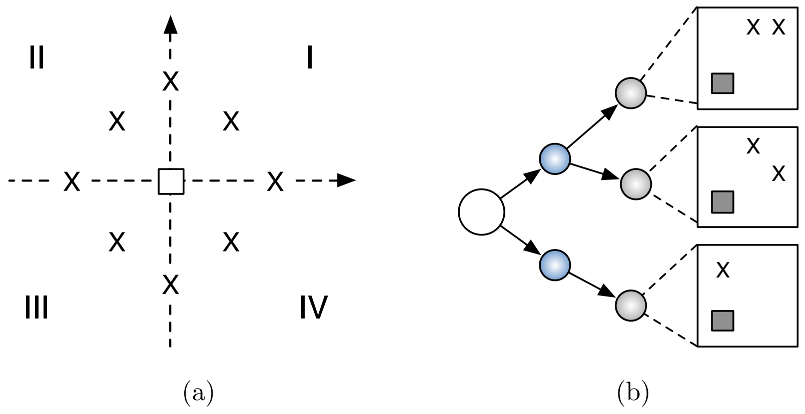


Figure 4.5: Example of signal identification: (a) Quadrant-based similarity vector; (b) Hierarchical isomorphic identification.

ning, these binding signal bits possess different numbers of pins which lead to a set of adjacent physical locations. Therefore, with the integration of signal groups, the algorithmic complexity increases with more possibilities.

To provide regular routes for bundled signals, we prefer to partition a provided signal group into a set of sub-groups, and deal with each sub-group as an individual routing object. In each object, every bit is able to acquire an equivalent topology and all its pins have the same SVs as the pins in other bits. That is to say, each pin is able to find its corresponding reflection from any other bit in the same object. After the routing flow, each bit in an object obtains an equivalent topology while for different objects in a signal group, they are preferable to share common topologies as much as possible.

With this objective, the partition strategy is illustrated as shown in Fig. 4.5(b). The input is one complete signal group represented by the white root node, and the output is a set of routing objects represented by the gray nodes, containing the bits owning the same similarity vectors for the pins. The squares in the right side provide the pin distributions of each signal bit belonging to an object. It is seen that all the bits are able to reach the same topology when their pins have the same relative direction. The methodology is intuitive but naive by calculating the similarity vector for each net pin, which would result in considerable calculation overhead. For those bits that are deemed to obtain different topologies based on their pin locations, we prefer to distinguish them as soon as possible without comparing each pin's similarity vector. Therefore, we adopt a hierarchical strategy based on the premise that the driver pins of various bits in the same group can be mapped mutually. In this way, we calculate the similarity vector of the driver for each bit at first. Those bits with different vectors are separated as blue nodes in the middle in Fig. 4.5(b). For the signal bits in the top blue node, their drivers have the same SV as $\{0,2,0,0,0,0,0,0\}$, although not all of them are able to obtain the same topology. It is easy to see equivalent topologies are infeasible for the bits whose drivers have different vectors. With this stage, the complexity decreases without traversing all the pins for each bit due to the fact that the number of pins in each direction in comparison to the driver is quite limited. Then, for those bits with multiple pins in one direction to the driver, as shown in Fig. 4.5(b), we only need to evaluate those pins for mapping. By taking the top blue node as an example, we can just compare the SVs of the pins in the first quadrant to the driver. Finally, the bits with the same SV for all the pins are combined as an object, and common topologies are encouraged for the objects from one signal group. In industrial designs, the signal groups are user-defined by referencing the specifications from many aspects, such as signal shielding, cell connection, etc.

4.3.2 Topology Generation and Evaluation

Before solving the signal routing problem, an efficient topology generation procedure is essential to provide candidate solutions. In this section, we propose a synergistic topology generation strategy for multi-pin connections which require equivalent topologies in one object and sharing topologies among objects in one group. It consists of three steps: backbone generation for every single object, equivalent routes generated for the bits based on the backbone and regularity evaluation among backbone topologies of various objects.

4.3.2.1 Backbone Structure Construction

After the isomorphic identification, a set of routing objects can be acquired from a group where all the bits can be routed with a topology, i.e. backbone structure. In essence, it is a topology prototype of all the bits in this object. The pins of one representative bit serve as the input information, and the output is a set of rectilinear connections of the pins and bending points with the same X/Y coordinates. Its formal definition is given as follows. To select a bit in the object, we choose one bit in the center region of an object and take its pins as the input. Since the identification stage distinguishes the bits sufficiently, a selected bit can be representative of all the bits in one object.

Definition 2 (Backbone Structure) *With a representative bit's pin locations, backbone structure is defined as a routing topology which every bit in the same object is able to use.*

For the topology generation, we extend the Batched Iterated 1-Steiner (BI1S) algorithm [35] based on an industrial flow. Since the topologies with many bends are not suitable for signal groups, the number of bending points is also an important index besides the wire-length. Considering that a backbone would affect all the bits in an object, it is essential to save wire-length while keeping as few bending points as possible. Therefore, a set of promising bending points should be selected for BI1S. It is known from Hanan grids that Steiner points should be located at the crossing points of input/output pins, which also conforms to bending points in our flow. Nevertheless, it is trivial to traverse all the internal edges connecting the pins and points, which may result in too many inferior candidates. Thus, we only extract the promising points and remove those resulting in long wire-lengths or complicated topologies. Then the selected points are saved into one queue with the priorities which indicate their potential wire-lengths and bending costs. To generate a set of topologies, we pick and insert the points from the queue to construct Steiner trees with the consideration of both wire-length and bending costs. Through the combination of pins and inserted points with a set of rectilinear connections (RCs), we are able to achieve a rectilinear Steiner tree. Then we select a non-inserted point from the queue with the highest priority to construct another tree. For each tree, at least one different bending point is adopted for the topology candidate. After visiting all the promising points at least once, we obtain an appropriate set of backbones on a 2-D plane for post-processing.

Since the objective of this procedure is to provide a set of topology prototypes for the objects, here we do not consider the required demands of tracks and the capacity constraints. For the demands through **G-Cell**s, the following equivalent topology generation phase will collect the topologies of the bits and calculate the total required tracks. With the exact topology of each bit, then the calculation will be accurate for reference. Besides, the possibly-occurring conflicts from different objects will be taken into careful consideration in Section 4.3.3. The reasons are as follows: Firstly, the routing layer has not been decided in the current stage, so the conflicts cannot be obtained due to the various capacities in multi-layer structure; secondly, as our optimal objective is to coordinate the routes from the objects while satisfying the capacity constraints, a comprehensive formulation with the global view of all these points is provided in Formula (4.3), which will be discussed in detail later.

4.3.2.2 Equivalent Topology Generation

Compared with classic escape routing, signal routing has more stringent constraints for the bits in a binding group: topology equivalence is required for

Algorithm 5 Equivalent Topology Generation

```
Input: Initial backbone $t_o$;
 1: Build LUT with $SV(p), p \in t_o$;
 2: Record $bt(t_o)$ with its connecting pins;
 3: for each bit $b \in$ do
 4:     Map $p \in b$ to $p \in t_o$;
 5:     while $\exists$ non-visited $bt(t_o)$ do
 6:         Select a non-visited $bt(t_o) \in t_o$;
 7:         Find $p_x(bt, t_o), p_y(bt, t_o)$;
 8:         Acquire $p_x(bt, b), p_y(bt, b)$ from map;
 9:         Determine $bt(b)$ based on $p_x(bt, b), p_y(bt, b)$;
10:         $bt(b)$ connect $p_x(bt, b)$, $bt(b)$ connect $p_y(bt, b)$;
11:     end while
12: end for
```

those bits in an object; common topologies among objects should be shared as much as possible. Section 4.3.1 describes how to partition a signal group into a set of objects, and a set of backbones is constructed for each object in Section 4.3.2.1. This section focuses on equivalent topology generation for an object according to each backbone. To achieve this objective, we refer to the similarity vector presented in Section 4.2.2. Through making sure the corresponding pin in the backbone for each bit, we are able to generate a topology same as backbone.

After the identification stage, all the bits in one object have the same SV for each pin. Thus, it is explicit to find the corresponding pin in backbone for each bit, and build a map to show this relationship. Based on a set of backbones generated beforehand, each equivalent topology is accompanied for each bit with the same connection of corresponding pins. To achieve this

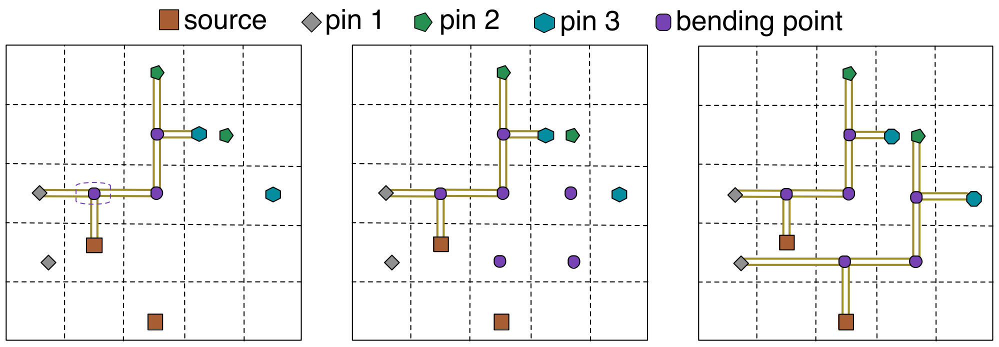


Figure 4.6: Equivalent topology generation example: (a) Pin mapping through similarity vector; (b) Bending points aligning; (c) Topology generation by connecting mapping pins and points.

objective, we first construct a look-up table (LUT) which captures the relative location of each pin in the corresponding bit. Taking the example in Fig. 4.6(a), pin_1 in the backbone will be mapped to the SV as {0,2,0,0,0,0,0,1}. During LUT construction, all the pins are traversed to record their SVs for further matching (line 1). Meanwhile, based on Hanan grids, bending points in the backbone are located with the same X/Y coordinates as its pins. Hence each bending point can be recognized easily through its neighboring connected pins (line 2). For instance, the circled bending point in Fig. 4.6(a) can be located by pin_1 and the source pin. Then we traverse each bit for topology generation in reference to each given backbone.

For each bit, through the LUT, each pin can be mapped to its reflection in the backbone according to its SV (line 4). For pin_1 of the non-routed bit in Fig. 4.6(a), due to the equivalence of its SV to pin_1 in the backbone, these two pins are mapped to each other. In the shown example, each pair of mapped pins is identified with the same shape in one color. After setting this matching relationship of pins, we start to build the topology by calibrating the bending points in each signal bit. During each iteration, a non-visited bending point is selected arbitrarily from the backbone, which has both horizontal and vertical connections to the pins (line 6). These connected pins, $p_x(bt, t_o), p_y(bt, t_o)$, are taken and utilized as the reference to align this bending point $bt(t_o)$ (line 7). Based on these pins, we obtain the corresponding matched pins in this signal bit, $p_x(bt, b), p_y(bt, b)$, with the assistance of the constructed map (line 8). Then, the bending point in the bit can be located with the same X coordinate as the vertical pin $p_x(bt, b)$, and Y coordinate as the horizontal pin $p_y(bt, b)$ (line 9). For the circled bending point in Fig. 4.6(a), the corresponding point will be aligned based on the X coordinate of the source pin, and Y coordinate of pin_1 in Fig. 4.6(b), and so on for the other bending points. By connecting the bending points with these neighboring pins with the same X/Y coordinates, an equivalent topology is able to be obtained (line 10). It is seen that with the LUT, the runtime of this algorithm is within $\mathcal{O}(|P_b||N_b| \log |P_b|)$, where $|N_b|$ represents the number of bits in an object, and $|P_b|$ represents the number of pins in a bit.

An explicit example is illustrated with three phases in Fig. 4.6. Fig. 4.6(a) provides the backbone and the mapped pins through LUT, while each corresponding pin is identified with the same shape in one color. With these mapped pins, the internal bending points are determined and aligned as shown in Fig. 4.6(b). Finally, an equivalent topology is given by connecting the pins and inserted points for the specified bit in Fig. 4.6(c).

Additionally, considering the existing multi-layer structure for current industrial designs, we develop a series of topologies with different layers based on each 2-D routing tree. For regularity, the horizontal and vertical trunks should be assigned on the same uni-directional layer; in the meantime, these trunks are preferred to be assigned on the neighboring layers in order to save the unnecessary via overheads.

4.3.2.3 Regularity Evaluation

Through the previous stages, equivalent topologies are guaranteed in each routing object. Nevertheless, since the signal groups are user-defined with pin locations in different directions, it is infeasible to enforce topology equivalence for all the objects in a given group. Therefore, we prefer to use a novel metric to quantify their topology differences. Considering that a backbone is able to represent the key structure for each object, it is explicit to take backbones into accounts for irregularity evaluation.

As described in Section 4.3.1, the pins in two bits can be mapped reciprocally according to their SVs when these two bits have the same number of pins, which could also be achieved through vertical sequences in FLUTE. However, for the bits with different numbers of pins, SV is able to target the most probable pin of another bit. To reach this objective, we adjust SV by incrementing the weight of driver pin which should be mapped to the drivers of other bits as expected. The weight is set to a value higher than the overall number of pins. Through this adjustment, the relative position of each pin to its driver is emphasized. Also, we calculate the SV for each bending point so that they can also be mapped to the pins or bending points of other topologies. By matching the pins/points with the closest SV in two topologies, t_1 and t_2, the regularity ratio is computed as in Equation (4.2). It is equal to the number of mapped rectilinear connections (RCs) formed by two mapped pins/points, NM_{RC}, divided by the minimum number of RCs in t_1 and t_2. As shown in Fig. 4.3(a), although the bottom object has one more bending point than the other, the topologies of these two objects are still regarded as similar topologies since this point can be mapped to the sink of the other object. Therefore, for this example, the ratio is set to 100% because both the number of mapped RCs and minimum number of RCs are equal to 1. In our algorithm flow, it is preferable to keep this ratio as high as possible to eliminate the dissimilar topologies. Since the denominator is more than or equal to the numerator in Equation (4.2), the highest value of the ratio is 1, which indicates that the given two topologies, t_1, t_2, share one topology.

$$\text{Ratio}(t_1, t_2) = \frac{NM_{RC}(t_1, t_2)}{\min\{N_{RC}(t_1), N_{RC}(t_2)\}}. \tag{4.2}$$

4.3.3 Mathematical Formulation

The mathematical formulation of Streak is provided in Formula (4.3). In the objective function, the first term is to calculate the total costs of all the objects, where $c(i, j)$ gives the cost of candidate x_{ij} of object i based on its wire-length and assigned layers. Since layering is taken into accounts, a post layer assignment stage can be saved to avoid potential sub-optimality. The second item is to enforce the routing of objects and M is a large penalty for those non-routed objects, whose s_i will be set to 1. Here S_c refers to the set of solution candidates, while S_o refers to the set of routing objects. To minimize the topology variance, we add the third item in Formula (4.3a). It helps to quantify the topology irregularity of any two objects in one group g, which is equal to the reciprocal of the regularity ratio. For two topologies which do not share any common rectilinear connections, a large number will be set to give the variance penalty but it should be smaller than M to ensure the first priority of signal routability. Meanwhile, for x_{ij} and x_{pq}, if they share any rectilinear connections but their assigned layers are not adjacent, a penalty proportional to the layer difference will also be assigned.

$$
\begin{aligned}
\mathbf{min} \quad & \sum_{(i,j)\in S_c} c(i,j)\cdot x_{ij} + \sum_{i\in S_o} M\cdot s_i + \\
& \sum_{(i,p)\in g}\sum_{(i,j)\in S_c}\sum_{(p,q)\in S_c} c(i,j,p,q)\cdot x_{ij}\cdot x_{pq} \qquad (4.3a)
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{s.t.} \quad & \sum_{(i,j)\in S_c} x_{ij} + s_i = 1, && \forall i\in S_o, && (4.3b)\\
& \sum_{(i,j)\in e_l} u_{e_l}(i,j)\cdot x_{ij} \le cap_{e_l}, && \forall e\in E, \forall l\in L, && (4.3c)\\
& s_i \ge 0, x_{ij} \text{ is binary}, && \forall i\in S_o, \forall j. && (4.3d)
\end{aligned}
$$

Meanwhile, constraint (4.3b) is to ensure that at most one topology is selected for each routing object; while constraint (4.3c) places the capacity limitation of each edge on different layers, i.e. cap_{e_l}. Due to the sharing topologies, we deal with a stringent edge capacity constraint for one edge can be utilized multiple times by several bits in an object concurrently, as shown in Fig. 4.3(a). Thus, we prefer to add one constant to provide the edge usage by the current topology, i.e. $u_{e_l}(i, j)$. Finally, with the constraints of both x_{ij} and s_i as binary variables, it is seen that this quadratic programming problem can be solved through integer linear programming (ILP).

4.3.4 Primal-Dual Algorithm

Although an ILP solver can be utilized to solve Formula (4.3), in real design it is not preferable due to its prohibitive runtime when a significant number of variables exist. We thus design a primal-dual algorithm to provide an efficient solution. With the generated topologies for each object, a fast and efficient flow is essential to make a sensible selection of candidates while satisfying the given requirements. A primal-dual algorithm is generally utilized for vertex covering problem, such as layer decomposition work in [90], which could also be applied in the routing flow by incrementing the dual variables accordingly. This section provides the details of how to solve the current routing flow through a primal-dual algorithm.

At first, we prefer to linearize the quadratic terms in Formula (4.3) for a primal formulation. Some previous works pre-define one of these two variables as a known value through an iteration-based framework [50,91], while [48] takes the quadratic terms through extensive Semidefinite Programming for more accuracy. Considering the properties of primal-dual, we search for the allowable minimum value of each term based on the states of x_{ij} and x_{pq}. Therefore, Equation (4.4) is utilized to provide a relatively accurate approximation:

$$\sum_{(i,p)\in g}\sum_{(i,j)\in S_c}\sum_{(p,q)\in S_c} c(i,j,p,q)\cdot x_{ij}\cdot x_{pq} \approx c'(i,j)\cdot x_{ij}, \tag{4.4}$$

where

$$c'(i,j) = \begin{cases} c(i,j,p,q), & \exists x_{pq}=1, \\ \min\{c(i,j,p,q)\}, & \forall x_{ij}\cdot x_{pq}\neq 0. \end{cases} \tag{4.5}$$

Since the primal-dual algorithm is a progressive flow through which x_{ij}s increase in a step-by-step manner, for a determined solution x_{pq} as 1, its combining cost with x_{ij} will be integrated with $c(i,j)$ as an additional cost. Nevertheless, if no solution has been decided for p, the minimum combining cost with any feasible x_{pq} will be considered as the cost. Here the feasibility refers to whether the combining topologies of x_{ij} and x_{pq} can still satisfy the current edge capacities. If not, this combination will be removed from the solution set. With this linear approximation, a dual problem ($\mathcal{DP}$) can be acquired as in Formula (4.6).

$$\mathcal{DP}: \quad \mathbf{max} \sum_{(i,j)\in S_c} \alpha_{ij} + \sum_{e_l\in E,L} cap_{e_l}\cdot\beta_{e_l} \tag{4.6a}$$

$$\mathbf{s.t.} \quad \alpha_{ij} + \sum_{i,j:e_l\in x_{ij}} u_{e_l}(i,j)\cdot\beta_{e_l} \leq c(i,j)+c'(i,j), \forall i,j, \tag{4.6b}$$

$$\alpha_{ij}\leq M, \qquad \forall i\in S_o, \forall j, \tag{4.6c}$$

$$\beta_{e_l}\leq 0, \qquad \forall e\in E, \forall l\in L. \tag{4.6d}$$

Formula (4.6) provides the dual form of Formula (4.3) with the linearizing item, which incorporates two types of dual variables: α_{ij} for constraint (4.3b) and β_{e_l} for constraint (4.3c). Based on the strong duality, the optimal solution for Formula (4.3) can be determined by satisfying the constraints in Formula (4.6). Therefore, we prefer to start with a primal infeasible but dual feasible solution set, and increment the primal solutions accordingly until a feasible solution is obtained.

The outline of the primal-dual algorithm is described in Algorithm 6, where the input is a set of routing objects with their candidate topologies. The initial primal solutions are set to 0 while keeping the dual variables also to 0 for their feasibilities (lines 1–2). Then the minimum required cost is calculated for each candidate to reach the upper bound of constraint (4.6b) (line 3). For each iteration, we check whether there still exist infeasible x_{ij}s and s_i, and the infeasible one with the minimum cost will be selected to increase its primal solution value (lines 5–6). Notably, here x_{ij} should be able to satisfy the current edge capacity constraints without any violations. Then the value of x_{ij} increases to 1 while s_i is kept to 0 due to the primal constraint. With the integration of solution x_{ij}, we update the available routing tracks of each edge passed by x_{ij} (line 8). Meanwhile, considering the decreasing usable tracks, some x_{pq}s become infeasible and their values are not allowed to rise. Thus, they can be removed securely without affecting the solution quality. For a specified object p, if all its x_{pq}s have been abandoned, s_p can be set to 1 (lines 10–12). Considering the existence of $x_{ij} \cdot x_{pq}$ in Formula (4.3), $c'(p, q)$ should be updated if it relates with x_{ij} (line 13). Since the physical characteristic of this quadratic term is the combining topology of x_{ij} and x_{pq}, $c'(p,q)$ should be re-calculated when some combining topologies are not available due to the reduced capacities. Through the search procedure, the sum of these dual variables keeps enhancing until an upper bound is reached by finishing all the solutions. During the whole process, edge capacity constraints are always held for infeasible solutions are already bounded beforehand.

Algorithm 6 Primal-Dual Algorithm

```
Input: A set of routing objects with its candidate set.
 1: Initiate primal solutions $x_{ij}, s_i$ to 0;
 2: Initiate dual solutions $\alpha_{ij}, \beta_{e_l}$ to 0;
 3: Calculate $c(i,j), c'(i,j)$ for each $x_{ij}$;
 4: while $\exists \sum x_{ij} + s_i = 0$ do
 5:     Search for a set of infeasible objects $i$;
 6:     Select $x_{ij}$ with the minimum $c'(i,j) + c(i,j)$;
 7:     $x_{ij} \leftarrow 1$, $s_i \leftarrow 0$;
 8:     Update $cap_{e_l}$ where $e_l \in x_{ij}$;
 9:     Remove infeasible primal solutions;
10:     if no feasible $x_{pq}$ for $p$ then
11:         $s_p \leftarrow 1$;
12:     end if
13:     Update $c'(p,q)$ for residual feasible solutions;
14: end while
```

4.4 Post Optimization

Section 4.3 provides a complete flow to coordinate topology selection and layering assignment for signal groups appropriately. Through the proposed Similarity Vector model in Section 4.2.4, the set of bits in a single object can

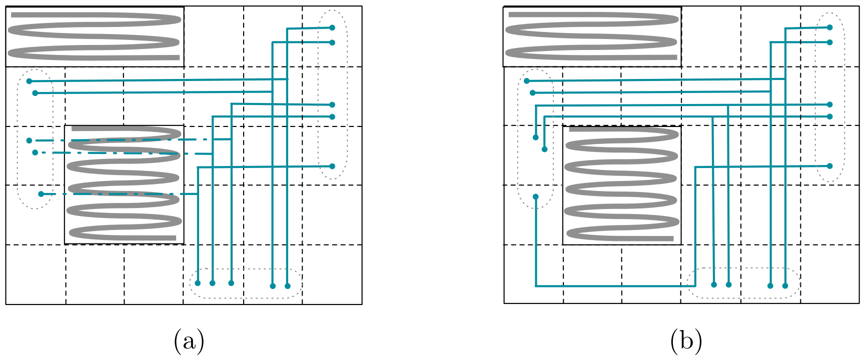


Figure 4.7: Example of blocked routing instance: (a) Routing of some bits blocked by obstacles; (b) Multiple topology selection for each cluster.

be determined to reach an equivalent topology. By generating a set of topology candidates for these signal bits, we are able to obtain the topology and layer assigning result for each object. Nevertheless, after the primal-dual flow, some signal objects may not be routed due to the very high number of bits for an object. That is to say, even for an object where its bits are able to reach the same topology, we may not be able to provide adequate routing resources for all the bits because of its required high widths. Therefore, it is imperative to provide further division for the non-routed objects as a post optimization stage so that more flexibilities are allowed, and the topology variance should also be controlled well among the bits for regularity.

To make it explicit, a blocked instance is given in Fig. 4.7(a), where the dashed circles signify the mapped pins for each bit. It is seen that all the bits can reach the same topology if there exists no such obstacle. Thus, to deal with this blocked issue, we prefer to allow further division for those bits so that more opportunities can be acquired to enhance the final routability.

Fig. 4.7(b) provides one possible solution, where three routing patterns are shown for all the bits instead of one. In this way, the blockage is bypassed from both the upper and lower direction without paying a high penalty of wire-length and topology variance. Generally, with a slight degradation of regularity, the existence of multiple clusters will offer more opportunities for those blocked objects. To ensure the effectiveness, it is essential to balance the trade-off between routability and design regularity.

Fig. 4.8 lists the outline of our post-optimization, which targets at improving the routability and refining topologies of signal groups. Instead of adopting common rip-up and reroute technique for nets, during signal routing we prefer to maintain the current topology and layer assignment solutions. The reasons are two-fold: Firstly, since one signal group contains many bits with regular routes and concurrent bending points, this increases the complexity of splitting those bits simultaneously. It is also hard to find another feasible routing space for re-routing the bits based on the limited track resources, and a domino effect can be caused by ripping up others continuously, resulting in unexpected distortion. Secondly, the proposed Primal-Dual flow considers the optimization of wire-length and topology regularity concurrently. Based on its closure to a global optimal result, it is intuitive to provide an incremental approach to take advantage of the residual resources without causing further disturbance. Therefore, we provide an outline of our post optimization flow in Fig. 4.8. For the signal groups to be routed, the preferable layers are predicted based on the congestion; and a bottom-up clustering methodology combines

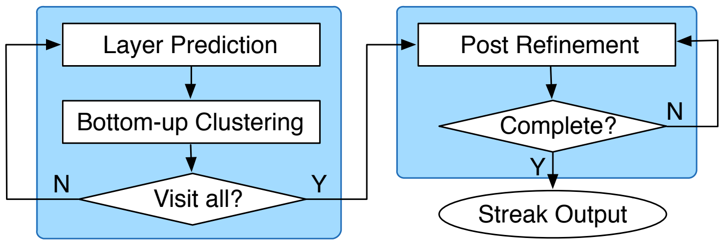


Figure 4.8: Post-optimization flow.

the bits while keeping legality for capacity constraints. This procedure continues until all these groups have been traversed. After this stage, we check if there exist source-to-sink distance deviation violations. If so, we will introduce appropriate twisting detours to refine the topologies. The details of each step are given in the following sub-sections.

4.4.1 Possible Layer Prediction

Different from traditional layer assignment works which behave after 2-D routing, we take layering into consideration before exact routing. Considering the occupied resources of those routed objects, it helps to narrow down the solution space efficiently by predicting the possible layers for the residual signals. Due to unidirectional routing on layers, it is required to select two layers favoring horizontal and vertical directions respectively, which offer the most available resources for the given group. Since the eventual routes have not been decided, a predictive methodology is utilized to give an estimation of track usages on each layer. Based on this approximation, the appropriate layers are selected with the least conflict values regarding the already routed bits.

To provide an estimation of track utilization, we take all the available topologies of the bits into accounts. For a 2-D edge e, its possible usage by a group g is calculated as follows,

$$u(e, g) = \sum_{b \in g} \sum_{t_j \in S_c(b)} \frac{1}{|S_c(b)|} \cdot u(e, t_j), \tag{4.7}$$

where $S_c(b)$ denotes the set of solution candidates for bit b, and $u(e, t_j)$ denotes whether this edge will be used by the jth topology candidate of bit b. Different from before, here we handle a non-routed bit as an individual object. Thus, even the bits belonging to one object can have different routing styles in order to reach a higher routability. Since the backbone generation stage provides a series of topologies for objects, every bit owns the same set of topologies according to its backbones, i.e. $S_c(b)$. Based on an assumption that each candidate has the same probability to be routed for bit b, we divide the summation of track usages from all the candidates by the candidate set size. This calculation is able to offer a close approximation of resource utilization by accumulating the bits in group g. Through considering all the bits' candidates, we obtain an estimated usage map of each concerned 2-D edge. Based on this usage map, we calculate the possible routing conflicts for each layer, as shown in Equation (4.8),

$$cf(l, g) = \sum_{e_l \in e} \max(u(e, g) - cap_{e_l}, 0), \tag{4.8}$$

where e_l is the corresponding 3-D edge on layer l for 2-D edge e in Equation (4.7), cap_{e_l} provides the available tracks for e_l, and $cf(l, g)$ is the estimated conflict value of routing group g on layer l. In each routing direction, the layer with the minimum conflict has the highest probability for group g to assign. Notably, this conflict value is based on an approximated congestion map which does not reflect the exact routing. In other words, even for a positive value, an overflow-free solution can still be achieved, and vice versa. Therefore, an efficient clustering and routing scheme plays an important role to avoid the conflicts while keeping regularity.

4.4.2 Bottom-up Clustering & Routing

In order to enhance the routability for signal groups, it is feasible to allow different topologies for the bits in one object. In this way, each bit can be handled as an individual for routing so that a higher routability can be achieved. Based on the layers obtained beforehand, it is important to search for appropriate routing solutions, which encourages both the routability enhancement and topology sharing among all the bits. Taking the example in Fig. 4.7(b), there are overall three routing styles, instead of only one in Section 4.3.1. Here one style corresponds to one cluster where the bits take the advantage of existing spaces to share a common topology for regularity. To achieve this, we propose a bottom-up clustering strategy to handle multi-bit routing intelligently.

The whole procedure is listed in Algorithm 7, where a set of non-routed groups serves as inputs. Initially, we produce the same set of topologies for the bits based on the backbones (line 1), and predict the layers with the highest probability for every group in both horizontal and vertical dimension (line 2). In each group, we construct a cluster for each bit so that they can be combined with others later (line 4). As a bottom-up clustering method, during each iteration, we ensure if there is a non-visited pair of clusters (line 5). If so, we will select a pair with the minimum achievable cost (line 6). To obtain this cost, we employ a similar way of cost calculation in Algorithm 6. For two clusters, if neither of them is routed, all the candidates will be traversed and the feasible solutions will be recorded with the corresponding cost; if one of them has been routed successfully, we will only take the non-routed cluster into accounts during the calculation. However, if no legal solution has been found, then a large penalty value will be counted. By considering all the available routes in two clusters, we acquire the minimum cost, which is set to the weighted sum of wire-length and regularity ratio. For the non-routed cluster, the candidate route with the best cost will be adopted (lines 7–9), and this pair of clusters will be marked as a visited one (line 10). Also, based on the routing styles, we check the regularity ratio to see if they share the equivalent topology (line 11). In this case, these two clusters should be combined further and the second one will be removed (lines 12–13). Through traversing and combining the cluster pairs appropriately, the bottom-up scheme explores the solution space efficiently with adequate options for the signal bits.

Algorithm 7 Bottom-up Clustering Algorithm

```
Input: A set of non-routed signal groups.
1: Topology candidate generation for bits in groups;
2: Possible layer prediction of signal groups;
3: for each group do
4:     Build one cluster $clus$ for each bit;
5:     while ∃ non-visited pair of clusters do
6:         Find a pair $(clus_1, clus_2)$ with the minimum cost;
7:         if $clus_1, clus_2$ not routed then
8:             Route with the minimum cost route;
9:         end if
10:        Mark this pair as visited;
11:        if Ratio$(clus_1, clus_2) = 1$ then                ▷ Equation (4.2)
12:            Merge two clusters $clus_1$ & $clus_2$ into $clus_1$;
13:            Remove $clus_2$;
14:        end if
15:    end while
16: end for
```

4.4.3 Post-Routing Refinement

The techniques above provide efficient routing control of signal bits through both top-down and bottom-up methodologies. Nevertheless, it still suffers the shortcoming that non-negligible source-to-sink distance variations result in possible signal malfunction. Different from classic bus routing, our framework deals with signal groups in which bits may have a different number of pins in various locations. Considering that the movement of one pin may disturb the other pins' conditions in a multi-pin bit, the problem becomes more complicated regarding signal routing. Meanwhile, topology regularity should also be taken into accounts throughout the whole procedure. Therefore, we present the following routing refinement methodology which shrinks the

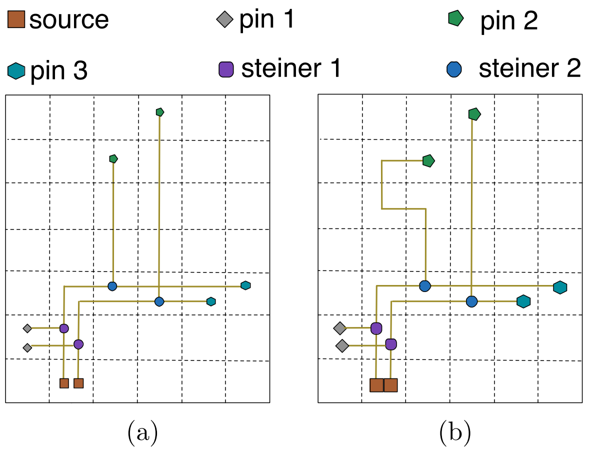


Figure 4.9: Example of bit-based source-to-sink distance adjustment: (a) Pin 2 violates the distance constraint; (b) Violation is fixed by introducing detour for pin 2.

distance difference with the consideration of regularity simultaneously.

As stated, it is likely that only a partial number of pins in one bit violate the source-to-sink distance constraint. Thus, our objective is to adjust the distances of those violating pins while trying to maintain the other pins' connections. An example is illustrated in Fig. 4.9, where two bits possess the same number of pins and each pair of mapped pins is signified with the same shape in one color. It is observed that the given two bits have the same topology and their regularity ratio is equal to 1. However, in Fig. 4.9(a), large distance difference exists for pin_2 but pin_1 and pin_3 share similar values without exceeding the threshold. To handle this, we split the topology of each bit into a set of rectilinear connections and only consider those connecting to pin_2. This is to say, only the connection from $steiner_2$ to pin_2 should be reconstructed. In this manner, not only does the problem size reduce, but the topology regularity is also under control by keeping the major topology. The resulting topology is shown in Fig. 4.9(b), where a twisting route is added for pin_2 to alleviate the distance violation.

Algorithm 8 Post Routing Refinement

```
Input: Set of violating signal groups $g_v$s;
1: Find violating bits $b_v$s and pins $p_v$s in $g_v$s;
2: Calculate current distance $dst_{p_v}$ for $p_v$, $p_v \in b_v$;
3: Calculate target distance $dst'_{p_v}$ for $p_v, p_v \in b_v$;
4: Acquire connection $conn(p_v)$ for $p_v, p_v \in b_v$;
5: for each group $g_v$ do
6:     for each bit $b_v$ do
7:         for each pin $p_v$ do
8:             Get starting point $sp_{p_v}$ of $conn(p_v)$;
9:             Get ending point $ep_{p_v}$ of $conn(p_v)$;
10:            if $conn(p_v)$ is horizontal then
11:                VerticalShift($conn(p_v), dst'_{p_v}$);
12:            else if $conn(p_v)$ is vertical then
13:                HorizontalShift($conn(p_v), dst'_{p_v}$);
14:            else
15:                for $x \leftarrow 0$ to $dst'_{p_v}$ do
16:                    $y \leftarrow dst'_{p_v} - x$;
17:                    VerticalShift($conn(p_v), x$);
18:                    HorizontalShift($conn(p_v), y$);
19:                    if $conn(p_v)$ is updated then
20:                        Break;
21:                    end if
22:                end for
23:            end if
24:            if $conn(p_v)$ is updated then
25:                Re-connect $sp_{p_v}$ and $ep_{p_v}$;
26:            end if
27:        end for
28:    end for
29: end for
```

The refinement details are provided in Algorithm 8, where a set of violating groups is taken as the input. First, we locate those bits which exceed the distance threshold. As the wire-length has been taken into consideration during the Primal-Dual flow, there is little space to reduce the maximum distance for its close to optimality. Thus, we select and signify the bits whose pins show much shorter distances compared to the other mapped pins (line 1). Then the current distances of these pins, i.e. dst_{p_v}, are recorded while the target distances, i.e. dst'_{p_v}, are also calculated (lines 2–3). Since there may exist only a few violating pins for a multi-pin bit, we traverse the original topology and locate these connections to be adjusted (line 4). After making sure the bits and corresponding pins to be handled, we will come to the details about allowing appropriate detours.

During detour production, our flow takes multi-layer capacity constraints into careful consideration to avoid further overflows. Thus, the expected twisting route is employed to complement the distance difference, i.e. $dst'(p_v) - dst(p_v)$, without any capacity violations. To exploit the residual available tracks, we allow the twisting route in four directions, i.e. left, right, lower and upper directions. As shown in Fig. 4.10, three possible types of horizontal shifting (left and right) can make up for the distance deviation, where the red (blue) points refer to the starting (ending) points of the given connection. Each type of them has an equal probability to be adopted as long as capacity constraints can be satisfied. Similarly, vertical shifting operations (lower and upper) are also performed when the starting and ending points have the same

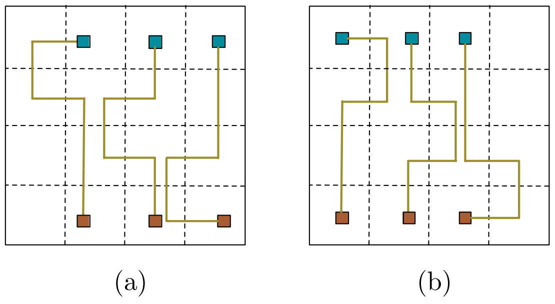


Figure 4.10: Example of horizontal shifting for source-to-sink distance matching: (a) Left shifting; (b) Right shifting.

Y coordinate. In the shifting methodology, as we focus on modifying the connections to the violating pins, topology regularity can still be maintained as much as possible.

Taking advantage of this shifting method, we adjust the distance of connections by traversing every violating pin p_v in the bit. Based on the acquired connection before, we ensure its starting point, sp_{p_v}, and ending point, ep_{p_v} (lines 8–9). Then we investigate whether the corresponding connection is an L-shape or a straight horizontal/vertical connection. For the horizontal connection, we perform vertical shifting to result in an additional distance in either upper or lower direction (lines 10–11). Similarly, horizontal shifting occurs to tune the vertical connection (lines 12–13). Nevertheless, for an L-shape connection, we are able to search for twisting routes in both two directions and obtain more choices for successful adjustment (lines 17–18). Due to the stringent capacity constraint, we traverse all the possible candidates in order to search for a legal solution. If it is found in both directions, then this searching procedure can be terminated to save the runtime overhead (lines

Table 4.1: Performance comparisons on 10nm industrial benchmarks.

Bench	#SG	#Net	Np_{max}	W_{max}	Manual Design		ILP				Primal-Dual			
					Route	WL (10^5)	Route	WL (10^5)	Avg(Reg)	CPU (s)	Route	WL (10^5)	Avg(Reg)	CPU (s)
`Industry1`	230	3722	2	75	100%	7.01	99.13%	7.30	99.13%	5.7	99.13%	7.30	98.12%	0.8
`Industry2`	492	12239	2	136	100%	17.24	99.59%	17.93	98.75%	107.6	99.59%	17.93	98.14%	2.0
`Industry3`	234	4402	2	70	100%	7.41	98.72%	7.34	96.94%	> 3600	98.72%	7.34	96.94%	1.2
`Industry4`	146	3446	2	147	100%	7.82	100.00%	7.79	97.72%	4.6	100.00%	7.79	97.72%	0.6
`Industry5`	587	11185	14	77	100%	15.00	99.32%	17.12	90.27%	> 3600	98.64%	17.23	89.85%	149.5
`Industry6`	409	7278	9	256	100%	11.25	99.27%	11.40	91.84%	> 3600	99.27%	11.40	90.98%	143.1
`Industry7`	171	4087	7	147	100%	12.40	100.00%	12.47	95.82%	54.7	100.00%	12.47	95.02%	1.2
average	-	-	-	-	100%	11.16	99.43%	11.62	95.78%		99.34%	11.64	95.25%	
ratio	-	-	-	-	**1.00**	**1.00**	**0.9943**	**1.041**	–	–	**0.9934**	**1.043**	–	–

19–21). Based on the updated connection, the previous one can be removed and re-connected to construct a new topology (lines 24–26). Since we build the new Steiner tree by traversing each violating pin, it should be noticed that the final topology should be a connected tree structure without any loops. After traversing all the violating pins in the certain bits and groups, the refinement stage stops and returns the improved routes. With the slight degradation of wire-lengths, the source-to-sink distance deviation can be controlled efficiently.

4.5 Experimental Results

We implemented the proposed Streak framework in C++, and tested it on a Linux machine with eight 3.3GHz CPUs. Meanwhile, we selected GUROBI [26] as our ILP solver. To evaluate its performance, we adopt seven industrial benchmarks with 10nm technology node: `Industry1`–`Industry7`. Each benchmark provides a set of signal groups which require further identification and synergistic operations as individual objects. The details of each benchmark suite are listed in the left part of TABLE 4.1. Here column "**#SG**" provides the number of signal groups, and column "**#Net**" corresponds to the total number of nets. With the existing multi-pin benchmarks, the maximum pin number of all the nets is listed in column "$\mathbf{Np_{max}}$", and the maximum bit number in each benchmark is also listed in column "$\mathbf{W_{max}}$".

4.5.1 ILP + Primal-Dual Performance Comparison.

Considering that few works handle signal routing of bundled bits with a varying number of pins in different directions, we obtain the manual designs by experienced designers from industry as shown in TABLE 4.1. Column "**Route**" provides the routability of all the groups, and column "**WL**" provides the wire-length measured manually. Since Streak also targets at synergistic routing for bits bundled in groups, an evaluation metric, "**Avg(Reg)**", is listed to show the average routing regularity for all the routed groups so that the routing synergy can be reflected without relying on the routability. Equation (4.9) explains how to calculate *Reg* for each group,

$$\text{Reg} = \frac{2 \cdot \sum_{t_i, t_p \in g} Ratio(t_i, t_p)}{N_o \cdot (N_o - 1)}, \tag{4.9}$$

where t_i, t_p represent the solutions from any two objects i, p in group g, and N_o is the number of objects in this group which should be larger than 1. Explicitly, for two topologies with more mapped RCs, the ratio will be higher but still smaller than 100%. In real design, the majority of signal groups are routed for regularity and wire-length improvement. In this manner, the signal bits in one group are encouraged to share the parallel routes, as the example shown in Fig. 4.1, and the parallel connections are assigned on the same layer. For the residual signal groups with complicated routing styles, the commercial tool, ICC [4], is called to accomplish the whole design, so the regularity ratio may not be guaranteed with the integration of this commercial tool. Finally, column "**CPU**" provides the runtime in seconds.

From the experimental results, it is shown that compared to manual design, around 4% wire-length overheads exist in average for seven benchmarks from ILP, where Primal-Dual provides a slightly higher value. To make a fair comparison, we calculate the total wire-length including both the routed and non-routed signal groups. For the non-routed groups, we estimate the wire-length based on Rectilinear Steiner Minimum Tree (RSMT) algorithm. Thus the reported wire-length represents the routing condition of a whole design. And both the average routability for ILP and Primal-Dual are more than 99%. Meanwhile, for the regularity rate, ILP and Primal-Dual can reach over 95% for two-pin signal groups, and keep more than 88% for test cases with multi-pin signal bits. Considering that a bit may have sinks in different directions to the driver, the regularity rate has already been constrained and this value is reasonable. Due to the capacity constraint in our flow, there is no capacity violation for all the benchmarks.

Additionally, the problem becomes complicated with both congestions and multi-pin connections. For a multi-pin design with low congestion, e.g.`Industry7`, ILP provides a good performance in short runtime. Nevertheless, for those with serious congestions, the ILP runtime is prohibitively long, so we terminate the

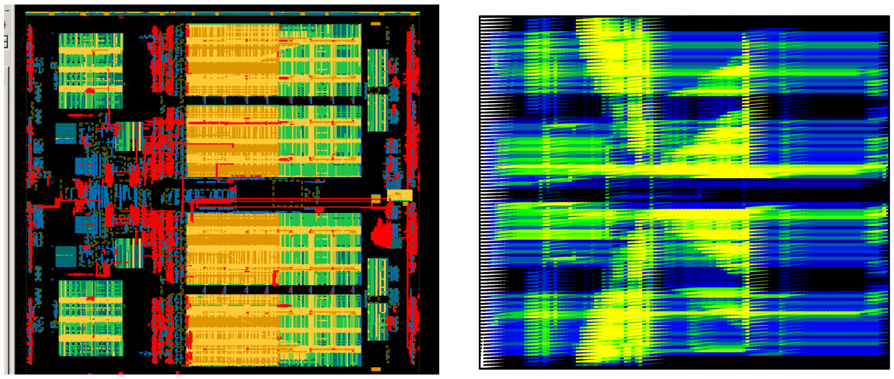


Figure 4.11: Routing congestion map for `Industry7`: (a) Manual design result; (b) Streak result.

Table 4.2: Performance comparisons of post optimization on 10nm industrial benchmarks

Bench	ILP	ILP + Post Opt					PD	PD + Post Opt				
	Vio(dst)	Vio(dst)	Route	WL (10^5)	Avg(Reg)	CPU (s)	Vio(dst)	Vio(dst)	Route	WL (10^5)	Avg(Reg)	CPU (s)
`Industry1`	12	0	100.00%	7.32	98.97%	10.1	12	0	100.00%	7.32	97.95%	4.7
`Industry2`	11	10	99.59%	17.98	98.54%	121.0	11	10	99.59%	17.98	97.93%	4.5
`Industry3`	10	0	99.15%	7.36	95.97%	> 3600	10	0	99.15%	7.37	96.00%	2.6
`Industry4`	6	2	100.00%	7.95	97.72%	4.7	6	2	100.00%	7.95	97.72%	0.8
`Industry5`	2	1	99.66%	17.16	90.25%	> 3600	2	1	99.15%	17.27	89.60%	198.2
`Industry6`	3	2	99.51%	11.40	91.30%	> 3600	3	2	99.76%	11.40	90.59%	145.7
`Industry7`	8	2	100.00%	12.66	95.82%	61.6	8	2	100.00%	12.66	95.02%	1.6
Average	7.4	2.4	99.70%	11.69	95.51%		7.4	2.4	99.66%	11.71	94.97%	
Ratio		**1.000**	**1.000**	1.000	1.000			**1.000**	**1.000**	1.002	0.994	

flow by setting a timing limit to 3600s. Comparatively, Primal-Dual is able to achieve comparable wire-length, routability and regularity rate much faster.

To provide a detailed comparison, we show the congestion densities for `Industry7` in Fig. 4.11 and `Industry6` in Fig. 4.12. Fig. 4.11(a) gives the congestion map from manual design, where the red regions indicate hotspots with overflows and lighter regions indicate more congested routing conditions. Both

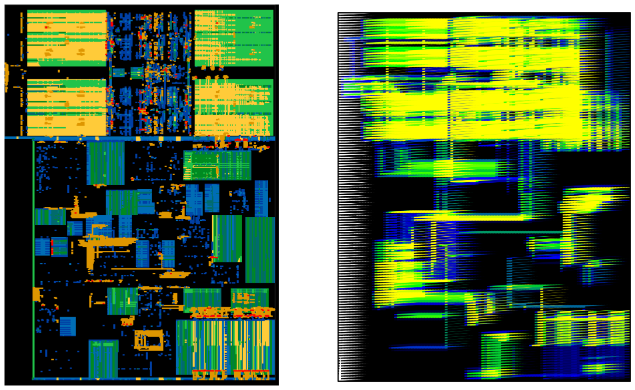


Figure 4.12: Routing congestion map for `Industry6`: (a) Manual design result; (b) Streak result.

with 100% as the routability, Streak in Fig. 4.11(b) allocates the routes in a balanced manner without any overflows. Meanwhile, regular routes can be observed with concurrent bending points. For a congested benchmark `Industry6` in Fig. 4.12, it is seen that the routes become complex for both manual and Streak result. Still, scattered overflow hotspots can be avoided by Streak efficiently. It is seen that with the slight sacrifice of routability, no overflow exhibits in Streak. Therefore, this comparison with manual designs proves the effectiveness of our tool to handle signal groups with synergistic routing styles.

To evaluate the algorithm scalability, we generate another large multi-pin benchmark based on `Industry2`, which offers the largest size among all two-pin testcases. During the generation, besides the existing two-pin connections, we insert some pseudo pins for the randomly selected groups so that the complicated routing styles can be obtained. Furthermore, the pseudo bits bundled in groups are also introduced to increase the potential conflicts. By

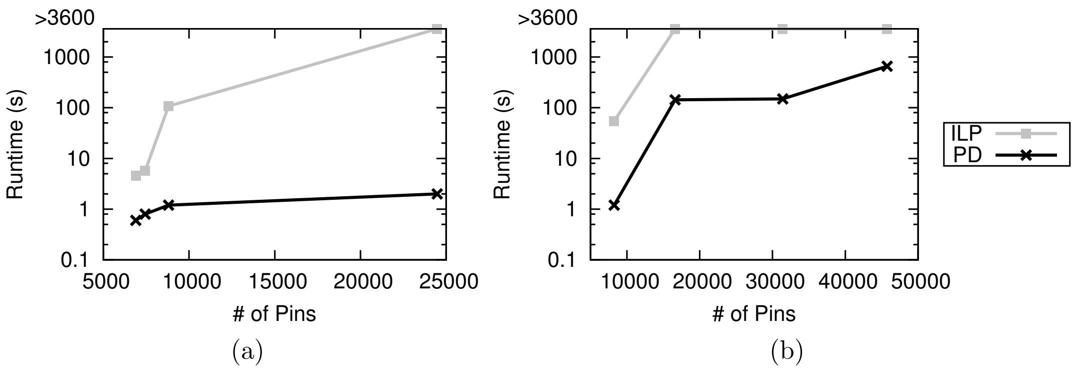


Figure 4.13: Performance comparison on algorithm scalability: (a) Two-pin benchmarks; (b) Multi-pin benchmarks.

conforming to the nature of signal routing, the pins of those generated bits are located in proximity. Then the scalability comparison in terms of total pins is provided in Fig. 4.13, where Fig. 4.13(a) shows the results of two-pin benchmarks, i.e.,`Industry1`–`Industry4`, and Fig. 4.13(b) shows the results of multi-pin benchmarks. The number of pins in the largest benchmark is given as the rightmost point in Fig. 4.13(b). For two-pin benchmarks, we observe that Primal-Dual provides a better scalability in comparison to ILP, especially with a larger scale. Meanwhile, the runtime of Primal-Dual increases in a small amplitude. Comparatively, a worse scalability is seen for both ILP and Primal-Dual in Fig. 4.13(b). It is understandable because multi-pin connections lead to more complicated routing styles compared to two-pin connections, and the conflicts from various signal groups are also aggravated. Still, Primal-Dual exhibits a better scalability than ILP, as expected, which verifies the effectiveness of Primal-Dual for both two-pin and multi-pin benchmarks. To improve the scalability of ILP, we may adopt varying sizes of **G-Cell**s iteratively to

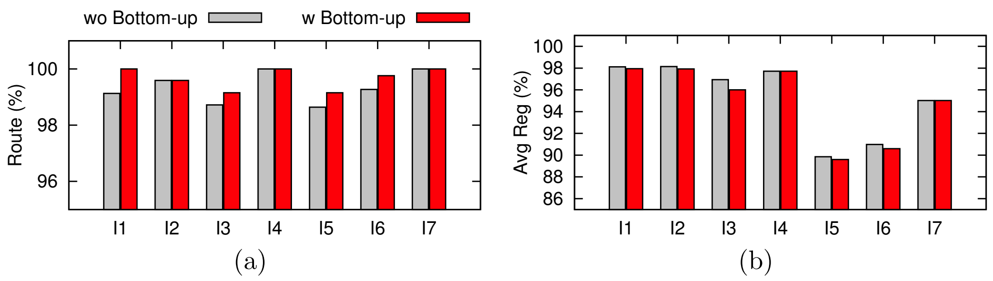


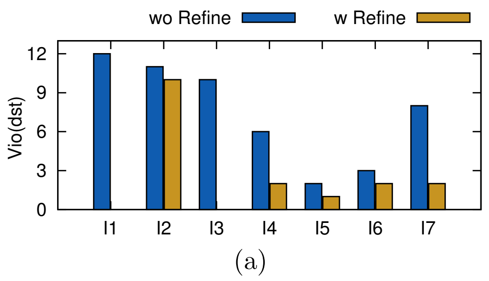
Figure 4.14: Performance comparison of bottom-up clustering: (a) Impact on routability; (b) Impact on average regularity.

solve the problem in a divide-and-conquer manner.

4.5.2 Effectiveness of Post Optimization

To prove the effectiveness of the post optimization, the results with and without this integration are listed in TABLE 4.2. Besides the columns illustrated above, we compare another metric, i.e. “**Vio(dst)**”, to evaluate the number of signal groups with the source-to-sink distance violation. To find an appropriate threshold value for each benchmark, here we set it to 50% of the maximum initial source-to-sink distance. The source-to-sink distance is the path length from the driver to the corresponding sink. In this setting, a few groups are marked as violated ones which exceed this given threshold and required to be adjusted through the refinement.

To demonstrate the effectiveness explicitly, we apply this post optimization to the solutions obtained from ILP and Primal-Dual, respectively. The numbers of violations before the post-optimization are listed in column 2 for ILP and column 8 for Primal-Dual. Both methods are with the same



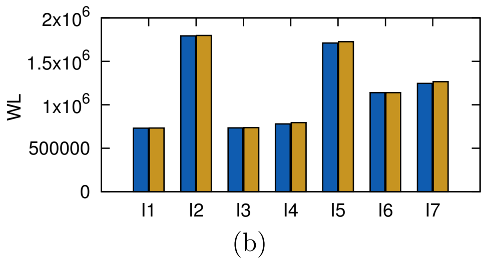


Figure 4.15: Performance comparison of post refinement: (a) Impact on violations; (b) Impact on wire-length.

violation number, which indicates that Primal-Dual is able to achieve the similar results as ILP. Meanwhile, the other columns provide the results of violations, routability, wire-length, average regularity ratio and runtime for both methods after the post optimization. As shown in TABLE 4.2, around 67% of violating groups can be fixed by introducing the extra detours, which verifies that the refinement stage has an efficient control of matching source-to-sink distances. Meanwhile, we observe that the routability values increase for both ILP and Primal-Dual, because the combination of layer prediction and clustering promotes more opportunities for signal bits to accomplish their routing. And the previous gap between ILP and Primal-Dual is also shrunk, which further validates its efficiency. Additionally, this post stage contributes to a slight degradation of wire-lengths compared with the initial results. Since the detours are induced to alleviate the violations during the post refinement phase, this increase of wire-length is expected and acceptable. Besides, the very similar wire-lengths are seen from ILP and Primal-Dual, and the regularity ratio becomes slightly lower for both methods. In essence, handling every bit as an object will lead to a worse regularity ratio; however, the bottom-up clustering methodology still takes topology variance into careful consideration and our post optimization targets at complementary routing for the residual groups without distorting the global planning, so the regularity ratio is in well control. Therefore, according to the results, a higher routability and a slightly lower regularity ratio can be achieved due to the combination of layer prediction and bottom-up clustering, and the distance violations can be reduced greatly through the post refinement. After the post optimization, we still reach the similar performance of ILP and Primal-Dual, which implies the routing consequence from the global view is respected sufficiently.

Besides, we perform two experiments by excluding the bottom-up clustering and the refinement stage to prove the validity in Fig. 4.14 and Fig. 4.15. Fig. 4.14(a) shows that the routability can be improved by around 0.3%, consistent with the objective of the clustering strategy. Meanwhile, although we search for the solution with the consideration of regularity in Algorithm 7, it still pays a slight penalty of regularity ratio, as shown in Fig. 4.14(b). Considering that more routing styles are enabled to enhance the routability, as illustrated in Fig. 4.7, a relatively lower regularity ratio is accepted. In addition, since the refinement stage targets to reduce the wire-length deviation, we list the number of existing violations and wire-length in Fig. 4.15. From Fig. 4.15(a), the number of violations can be controlled efficiently through the proposed refinement, but it suffers from the wire-length penalty in Fig. 4.15(b), which results from the twisting overheads for distance matching. Because we only allow the necessary detours, the total overhead is negligible.

4.6 Summary

In this chapter, we have proposed a set of algorithms to generate synergistic topology for on-chip signal groups. First signal bits with distinctive connections are identified and then combined as routing objects with equivalent topologies. A mathematical formulation targets at wire-length and routability optimization while controlling the topology differences, while a fast flow matches a close quality with manual design and ILP results. To improve the routability of signal groups with more flexibilities, a post-optimization stage allocates appropriate routes for each bit with the control of regularity. A post-routing refinement strategy follows to decrease the source-to-sink distance deviation of signal bits. The results show that our synthesis tool is able to provide efficient routing solutions with full legality and reasonable congestion map.

Chapter 5

Optical-electrical Power-efficient Route Synthesis

5.1 Introduction

The previous chapter has provided an introduction of signal routing in the electrical field. As interconnect delay becomes a bottleneck towards timing closure, research efforts have shifted to explore efficient interconnect to replace electrical wires. Due to the inherent dominance of bandwidth-distance-power properties, optical communication based on chip-scale optical-electrical systems emerges as a promising alternative [78]. Thus this chapter presents the power-efficient route synthesis flow by incorporating optical and electrical interconnects smoothly. Since recent fabrication techniques enable the on-chip integration of nano-scale devices with a high density, nanophotonic interconnects show great potential in unleashing the bandwidth limitation for memory access and processor communication.

With the increasing trend of on-chip communication density, Wavelength Division Multiplexing (WDM) exhibits the potential of offering high bandwidth with controllable overheads. During the transmission, signal quality can be affected by various losses at the receiving side. And EO/OE conversion power overheads should also be considered carefully [77]. Thus, it is desirable to propose an efficient optical-electrical co-design engine by offering optical and electrical connections for local and global communication, respectively. As shown in Figure 5.1, it consists of two layers: the bottom one for copper wires and the upper one for optical interconnects. The yellow arrows denote the electrical wires, and the blue arrows denote the optical interconnects for remote connections. Based on this infrastructure, an intelligent framework is essential to direct the distribution of electrical and optical wires for on-chip communication.

There are a few works dealing with the integration of optical interconnects onto on-chip designs. Some works provide physical design of on-chip optical interconnect: Ding et al. [23] proposed the first optical router for low power consumption, and employed WDM architecture to reach high density/capacity during global routing while ignoring the splitting loss [22]. As the first placement-and-routing tool for optical Network-on-Chip(NoC)s, Boos et al. [9] balanced the trade-off between propagation and crossing losses but limited to two-pin connections. Besides, optical NoCs were also designed to enhance its resilience to physical variations, such as environmental temperature and manufacturing instability [22, 63]. To increase the potential bandwidth, communication parallelism was emphasized in [68] through a formal methodology. Very recently, with the proposal of LED-driven wires, an automatic place-and-route flow enabled the replacement of electrical wires with on-chip laser sources [41].

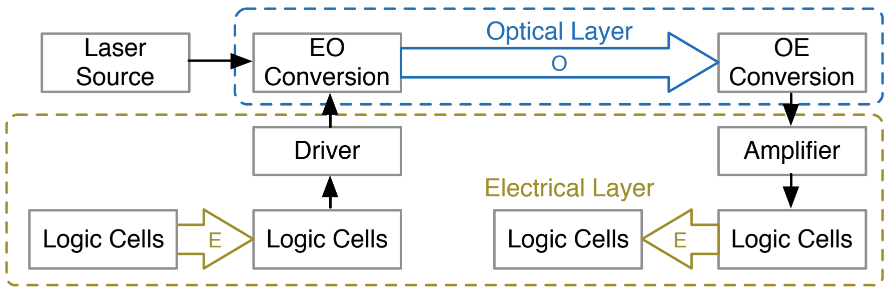


Figure 5.1: Block Diagram of Optical-Electrical On-Chip Design.

With the high-bandwidth demands of on-chip communication, it is desired that optical interconnects collaborate with electrical counterparts smoothly [8]. Very few of previous works provide a comprehensive routing flow for optical-electrical co-design of on-chip multi-pin signals. Without loss of generality, optical configurations are deployed for the distant connections. Nevertheless, signal transmission may suffer from non-negligible optical loss, resulting in the potential malfunction. And the splitting loss also aggravates the resulting loss, which plays an important role but is neglected in the previous optical physical design works. By introducing the reasonable optical-electrical configurations, we can make a better trade-off between the optical loss and power consumptions. Therefore, it is worthwhile to devise an efficient routing flow where optical connections can collaborate with electrical wires seamlessly.

In this chapter, we propose a power-efficient routing synthesis flow for optical interconnects to be integrated with electrical wires. The results have been presented in [52], and the contributions are summarized as follows:

- With splitting loss into consideration, optical-electrical co-design routes are derived for power and loss optimization;

- Mathematical formulation targets power minimization while satisfying the detection constraints, and a Lagrangian-Relaxation-based algorithm is presented for speed-up;
- Network-based algorithm assigns the optical connections for sharing WDMs with the control of distance.

The remainder of this chapter is organized as follows. Section 5.2 presents the overview of our framework and adopted models. Section 5.3 describes our routing procedure, presents a mathematical formulation to optimize power consumption while satisfying the detection constraints, and a Lagrangian-Relaxation method benefits the runtime. Section 5.4 presents the WDM assignment. Section 5.5 reports the experimental results and followed by the summary in Section 5.6.

5.2 Preliminaries

In this section, we provide the overview of our proposed framework, and illustrate the adopted model and methodology, based on which a problem formulation is given.

5.2.1 Overall Flow

To provide a clear view of our framework, an outline is shown in Figure 5.2. Firstly, a processing step partitions signal groups into a set of hyper

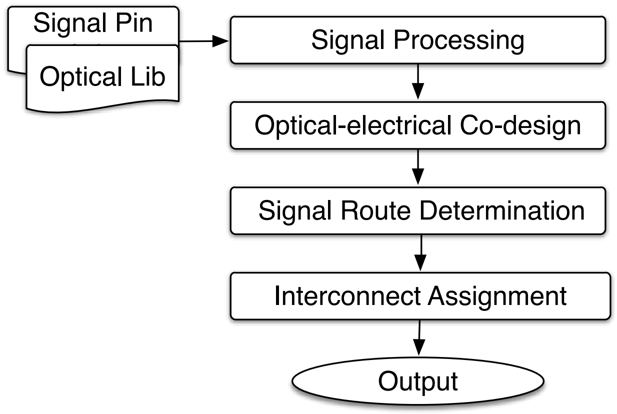


Figure 5.2: OPERON Flow.

nets in a top-down manner while clustering the neighboring electrical pins from a bottom-up view. Secondly, we devise the optical-electrical co-design routes for each hyper net. With the acquired solution candidates, a mathematical formulation is given for solution determination; to avoid the potential runtime overheads, we adopt a fast algorithm for speed-up. Then the WDM placement is performed and an assignment procedure follows to allocate optical connections onto nearby WDMs. We finally acquire the electrical-optical co-design solutions.

5.2.2 Optical Device Model

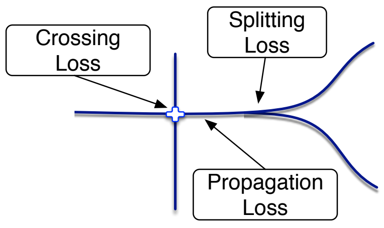
With the advanced fabrication technologies, emerging optical devices have been promoting efficient on-chip communication. The WDM infrastructure offers multi-channel routing among physical locations, which is a promising alternative for high-speed data communication. Therefore, it provides great potential for routing on-chip parallel connections, without crosstalk issues between different channels.

In comparison to electrical wires, WDMs contribute to high bandwidth and low power consumption for data propagation. Nevertheless, the EO/OE conversion leads to non-negligible power consumption due to the extra driver and amplifier configuration, which are usually neglected during routing optimization. Hence we provide the optical power, p_o, by using WDMs,

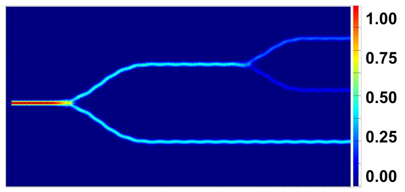
$$p_o = p_{mod} \cdot n_{mod} + p_{det} \cdot n_{det}, \tag{5.1}$$

where n_{mod}, n_{det} represent the number of employed modulators and detectors, and p_{mod}, p_{det} represent the unit power cost of modulators and detectors [77].

Besides the power cost, the optical loss along the WDM should also be taken into careful consideration. As shown in Figure 5.3(a), the loss mainly consists of propagation loss, crossing loss, and splitting loss. The first two kinds of loss are respectively in proportion to the total WDM length and the number of crossing occurrences. Notably, the splitting loss has usually been neglected in the previous work, which, however, turns out to be one of the major sources of loss for on-chip optical routing [95]. The splitting loss happens whenever an input light source splits into multiple light sinks. The expected splitting loss in dB for each splitting is calculated to be $10 \cdot \sum log(n_s)$, where n_s is the number of splitting arms. As seen in Figure 5.3(b) that shows the power distribution of two cascaded 50-50 Y-branch splitters based on the simulation, each reduces the input light power into one half on the output sides. Thus,



(a)



(b)

Figure 5.3: Optical model illustration. (a) Loss model for on-chip optical routing; (b) Simulation of the normalized power loss in Y-branch splitters.

the loss is calculated as follows,

$$loss = \alpha \cdot WL + \beta \cdot n_x + 10 \cdot \sum log(n_s), \tag{5.2}$$

where WL is the WDM length, n_x is the number of crossings, and n_s is the number of splitting arms. α and β are the physical parameters for the propagation and crossing loss.

5.2.3 Proposed Signal Model

The on-chip integration of optical interconnect provides the opportunity for signals to be routed in parallel routes. In current industrial designs, the performance-critical signal bits are bound together for data communication between logic cells and memory interfaces, etc. Figure 5.4(a) gives an example of signal routing on a 2D optical layer, where each signal group is identified with one single color. For the grey signals, they are clustered into two hyper nets because the number of total bits exceeds the WDM capacity. Different from Manhattan routing based on electric wires, the optical scheme

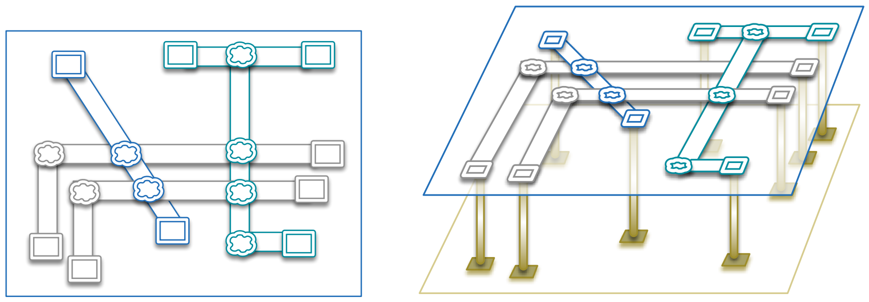

(a) (b)

Figure 5.4: On-chip signal model. (a) Signal routing on 2D optical layer; (b) Signal routing on 3D optical-electrical architecture.

allows routing in any direction, which can benefit the overall wire-length. With the EO/OE conversion deployment, the optical interconnects are coupled to the bottom electrical layer, as shown in Figure 5.4(b). This architecture offers us the flexibility of optical-electrical co-design for on-chip signals.

5.2.4 Problem Formulation

Based on the proposed flow and optical model discussed in the preceding section, we define the proposed optical-electrical route synthesis (OPERON) problem as follows:

Problem 4 (OPERON) *Given signals bundled in groups with the pin locations, our framework determines the routing topologies and optical-electrical configurations for each signal group so that the total power consumption can be optimized while the detection constraints are satisfied.*

5.3 Algorithms

In this section, we will elaborate the procedure of signal routing through optical-electrical cooperation for power efficiency. With the constructed hyper nets, route candidates are generated with co-design optimization. A mathematical formulation guides the routing concurrently, which follows with a speed-up algorithm.

5.3.1 Signal Processing

Before deriving the route solutions, a processing procedure is required for creating the pseudo pins and hyper nets of the signal bits. Considering that the pin distances may vary significantly for one bit, it brings the necessity of constructing the pseudo pins to represent the neighboring electrical pins. Besides, signal bits can be bundled together as a hyper net whose pins are located in adjacent locations. Therefore, we employ the following clustering methodologies for hyper net construction.

5.3.1.1 K-Means-based Clustering

Since the capacity, i.e., the number of allowable channels, of one WDM is limited according to the current fabrication technologies, we should determine how to cluster the bits to satisfy the capacity constraint. Thus, for a given signal group, we first check if the number of bits is above the capacity. If so, then we partition this group based on the K-Means strategy, which is widely used in clustering for its effectiveness [56].

Due to its top-down nature, the signal bits are divided into K clusters and K is the quotient of the total bit and the capacity value. Since K-Means itself cannot guarantee the cluster size during the solving process, we extend the K-Means strategy by checking the size in every iteration: if a cluster violates the capacity constraint, the additional bits will be assigned to the second closest one, and so on. As K clusters are adequate for accommodating all the bits, the neighboring bits can be located in one cluster with the decreasing distance variance. When the variance becomes lower than a given threshold, this iterative flow will stop. There may be a few empty clusters without any assigned bits, which will be removed afterward.

5.3.1.2 Hyper Net Construction

Based on the K-Means solution, we construct the hyper net to represent all the bits in a cluster. By replacing the set of individual nets with a single hyper net, the whole problem size can be reduced. We build up the pseudo pins for the hyper net and determine their corresponding electrical pins.

For one cluster containing a set of neighboring bits, each electrical pin itself is initialized as a hyper pin. Then we adopt a bottom-up clustering strategy for shrinking the hyper pins. During each iteration, a pair of hyper pins is selected with the minimum Euclidean distance. And we check if their physical distance is below the pre-specified threshold: if so, then these two hyper pins will be combined with the updated gravity center; otherwise, the clustering will return with the finalized set of hyper pins. As each hyper pin contains a set of electrical pins eventually, we should ensure the number of connections between the hyper pins. Based on the constructed hyper nets and pins, we perform routing design and synthesis in the following sections.

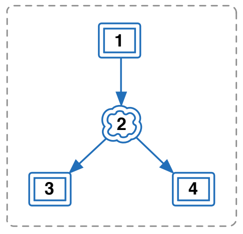
5.3.2 Optical-electrical Route Co-design

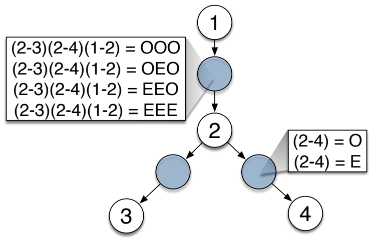
With the assistance of processing, we acquire a set of hyper nets with the corresponding pins for connection. Here we discuss how to combine optical and electrical design in a coordinated way. Besides supporting optical interconnects for distant connections as the previous works [22], our co-design methodology provides a more systematic analysis of power consumption and optical loss.

Before the optical-electrical development, the sets of optical route candidates are generated as the baselines for co-design processing. Here we choose to extend the Batched Iterated 1-Steiner (BI1S) algorithm which provides the flexibility of Steiner point selection. By sorting the Steiner points with the induced propagation and bending cost, we can acquire various baselines by visiting different points. Additionally, compared to Manhattan routing of electrical wires, optical interconnects are able to route in any direction, as shown in Figure 5.5. Thus, the rectilinear connections are not mandatory for optical interconnects, which offers more feasible baseline topologies. After obtaining the baselines, we present the optical-electrical co-design scheme for power and loss optimization.

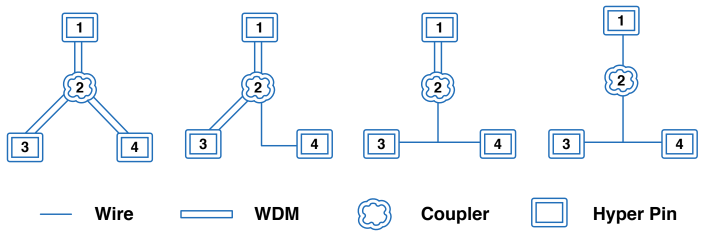
As the baseline is a tree topology, in essence, we take this advantage to develop a list of co-designs, each recorded with competitive optical loss and power cost. Taking Figure 5.5(a) as a baseline instance, we visit each connection between the hyper pins and Steiner points and decide which to employ optical interconnections. Inspired by the classic buffer insertion algorithm, we derive the promising co-designs in a bottom-up manner so that the trade-off between loss and power can be balanced.

The algorithm flow is illustrated in Figure 5.5(b). By traversing $node_3$ and $node_4$ in the bottom level, each interconnect can choose to route through optical WDMs or electrical wires, and the internal node records each solution with the specific power and optical loss. The optical power is calculated as in Eq. (5.1), while the electrical power is in proportional to the wire-length. For the optical loss, we are able to calculate the exact propagation and splitting loss and approximate the crossing loss based on the optical baselines. If the optical interconnect between $node_2$ and $node_4$ causes both higher loss and power costs compared to that between $node_2$ and $node_3$, then the former candidate turns to be an inferior solution to be pruned beforehand, and we come to the upper level for further judgments. The internal node between $node_1$ and $node_2$ accumulates all the possible solutions, and the resulting power cost and optical loss are re-calculated. As a redundant solution is removed, there remain four solutions with different configurations. Figure 5.5(c) lists all the finalized solutions. It can be seen that the third candidate can save the OE conversion overheads by implementing the bottom branches through electrical wires. After traversing all the nodes, we acquire the optical-electrical solutions


(a)



(b)



(c)

Figure 5.5: Optical-electrical co-design example. (a) Hyper net topology; (b) Dynamic programming based co-design scheme; (c) Corresponding optical-electrical solution candidates.

while eliminating the non-competitive alternatives. The runtime complexity of this procedure is within $\mathcal{O}(|N_c||d|)$, where $|N_c|$ is the number of connections between the hyper pins and Steiner points, and $|d|$ is the depth of the tree in Figure 5.5(b).

5.3.3 Mathematical Formulation

To obtain the optimal solution for each object, the mathematical formulation is given in Formula (5.3). The objective is to minimize the total power overheads for all the hyper nets H. The set of optical-electrical solution candidates is denoted as H_{sol}, which consists of both optical-electrical co-design and pure electrical routes for the hyper nets. The first item, a_{ij}, represents the j-th solution candidate for hyper net i, and $p_{oe}(i,j)$ gives the corresponding power costs with OE/EO conversions, as described in Section 5.2.2. Additionally, a_{ie} represents the electrical route alternative of hyper net i through electrical wires, as the fourth candidate in Figure 5.5(c), and $p_e(i)$ represents the power consumption cost of a_{ie}.

$$
\begin{aligned}
\textbf{min} \quad & \sum_{(i,j)\in H_{sol}} p_{oe}(i,j)\cdot a_{ij} + \sum_{i\in H} p_e(i)\cdot a_{ie} && \text{(5.3a)}\\
\textbf{s.t.} \quad & \sum_{(i,j)\in H_{sol}} a_{ij} + a_{ie} = 1, \quad \forall i \in H, && \text{(5.3b)}\\
& \sum_{(m,n)\in H_{sol}} l_x(i,j,m,n,p)\cdot a_{ij}\cdot a_{mn} + l_s(i,j,p)\cdot a_{ij} \\
& \qquad + l_{spl}(i,j,p)\cdot a_{ij} \le l_m, \quad \forall p \in P(a_{ij}), \quad i \in H, && \text{(5.3c)}\\
& a_{ij}, a_{ie} \text{ is binary}, \quad \forall i \in H, \quad \forall j. && \text{(5.3d)}
\end{aligned}
$$

Meanwhile, constraint (5.3b) denotes that one and only one solution candidate can be selected for each hyper net: if no appropriate optical-electrical co-design route is found, then this hyper net will be handled with electrical wires. Also, based on the optical inherent constraints, the light intensity should be strong enough to be detected at the receiver side. Hence, constraint (5.3c) guarantees that the total source-to-sink loss on path p should be lower than the maximum loss, l_m, and p corresponds to one source-to-sink path in a co-design candidate a_{ij}. $l_x(i,j,m,n,p)$ refers to the crossing loss on path p resulting from the intersections between candidates a_{ij} and a_{mn}, while $l_s(i,j,p)$ and $l_{spl}(i,j,p)$ are the propagation loss and splitting loss of path p, respectively. Finally, with the constraints of both a_{ij} and a_{ie} as binary variables, it is seen that this quadratic programming problem can be solved through Integer Linear Programming (ILP). Due to the existing terms of a_{ie}s, a feasible solution can be guaranteed for all the hyper nets.

Nevertheless, solving an ILP would lead to prohibitive runtime overheads. Thus the speed-up technique is adopted to condense the solution space without sacrificing the performance. To reduce the number of variables, we can remove those crossing variables belonging to the pair of hyper nets with non-overlapped bounding boxes. By this setting, we can control the search space without performance degradation.

5.3.4 Lagrangian Relaxation-based Algorithm

Since solving the ILP even with the reduced variables would still be time-consuming, we re-formulate this problem in a more efficient way. Therefore, we propose a Lagrangian Relaxation-based (LR) approach, which relaxes the constraint (5.3c) into the objective function. The relaxed formula is shown in Formula (5.4) with the Lagrangian Multiplier (LM) λ_p for each path p. Since l_m is a constant value, we remove it in Formula (5.4a).

$$\begin{aligned}\textbf{min}\quad & \sum_{(i,j)\in H_{sol}} p_{oe}(i,j)\cdot a_{ij} + \sum_{i\in H} p_e(i)\cdot a_{ie} \\ & + \sum_{p\in P(H_{sol})}\sum_{(m,n)\in H_{sol}} \lambda_p \cdot l_x(i,j,m,n,p)\cdot a_{ij}\cdot a_{mn} \\ & + \sum_{p\in P(H_{sol})} \lambda_p \cdot (l_s(i,j,p) + l_{spl}(i,j,p))\cdot a_{ij} \qquad (5.4a) \\ \textbf{s.t.}\quad & (5.3b), (5.3d).\end{aligned}$$

To resolve an LR-based algorithm, we update the LMs iteratively to explore the solution. Meanwhile, the quadratic terms can be linearized based on the last iteration, which empirically works well [50, 91]. Thus we employ this approximation method to capture the quadratic terms:

$$a_{mn} \cdot a_{ij} \approx a'_{mn} \cdot a_{ij} + a_{mn} \cdot a'_{ij}. \tag{5.5}$$

By substituting the linearization terms, it is seen that the routing optimization becomes a weighted sum of a_{ij}s and a_{ie}s. With the fixed set of LMs and constraint (5.3b), we search for the solution in each iteration. The pseudocode of the LR-based algorithm is shown in Algorithm 9. Initially, the LMs are set to a value proportional to the p_e (line 1) while the power cost is calculated for the candidate solutions (line 2). And the propagation and splitting loss, $l_s(i,j,p), l_{spl}(i,j,p)$, are also calculated for each optical-electrical route candidate (line 3). During the Lagrangian optimization, we traverse the hyper nets in each iteration and select the candidate with the best sum of weights, including both its inherent power and LM penalty costs (line 5). Based on the acquired solutions, we check the detection violations for each source-to-sink path (line 6). Then the LMs are updated at the end of each iteration according to the violations (line 7). To ensure the algorithm convergence, we update the LMs through a sub-gradient methodology. This procedure continues until a convergence is reached. The converging criteria are set as follows: the decrease in both power costs and violations reach a pre-defined ratio, or the iteration number is over 10.

Algorithm 9 LR-based Algorithm

```
Input: A set of hyper nets with candidates $a_{ij}, a_{ie}$.
1: Initialize LMs $\lambda_p$s;
2: Calculate the power costs for $a_{ij}, a_{ie}$s;
3: Calculate the loss $l_s(i,j,p), l_{spl}(i,j,p)$ for $a_{ij}$s;
4: while no converge do
5:     Select the candidate with the best weight;
6:     Calculate violation values for paths;
7:     Update $\lambda_p$s based on violations;
8: end while
```

5.4 WDM Assignment

After the last procedure, each hyper net has obtained its topology consisting of point-to-point connections. Similar to electrical wires, the distribution of optical connections should also be controlled. The WDM offers multi-channels for parallel routing, but its capacity is constrained. Thus, the assignment is able to utilize WDMs efficiently without disturbing the previous result. In this section, we describe the WDM placement and the assignment procedure through a network model.

5.4.1 WDM Placement

Before the assignment of connections, we perform the WDM placement to initialize their locations. An intuitive way is to place one WDM for each connection. However, this would lead to an unnecessarily high volume of WDMs for which can be shared by the hyper nets propagating in parallel; Also, we control the distance to be above dis_l between two nearby WDMs to avoid crosstalk. Thus, the placement procedure not only controls the number of WDMs, but also conforms to the distance bound.

Since the placement of vertical and horizontal WDMs follow the same way, for brevity we only discuss the horizontal WDMs. Initially, the horizontal connections are collected and sorted in an ascending order of their y-coordinates. Then we traverse each connection sequentially to determine the WDMs' locations. The first WDM is placed in the same location as the first connection, and recognized as the current WDM. For the next connection, we check whether the current WDM has enough capacity. Also, their distance should be below the distance dis_u to avoid causing the disturbances. If both of conditions are met, the connection will be assigned to the current WDM; otherwise, we place an additional WDM and turn to the next connection. After visiting all the connections, an adequate number of WDMs will be obtained for assignment. As the placement enables the combination of multiple connections in cases, the number of WDMs is already well controlled. It shall be noted that after the placement there may exist some congested regions where two nearby WDMs are within dis_l distance. To avoid these cases, we check those violating regions and adjust the WDMs' locations in a one-by-one way for legalization.

5.4.2 Network-flow Based Assignment

After initializing the WDM placement, we will assign the optical connections while removing the idle WDMs. Previously, a number of WDMs are utilized to accommodate the signal bits in a sequential manner, which does

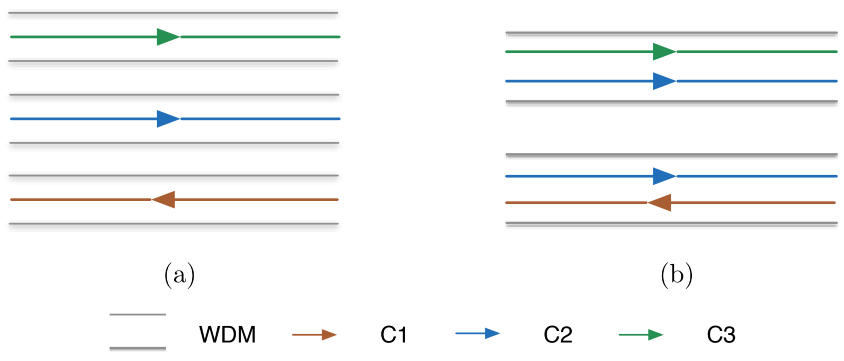


Figure 5.6: Example of WDM Placement. (a) Initial WDM placement for three connections; (b) WDM placement after the assignment.

not exploit the WDMs in a global view. This brings out the necessity of re-assigning the connections concurrently. As we observe in Figure 5.6(a), for three connections marked in different colors, we adopt three WDMs if each connection contains 20 bits and the capacity is 32 as set in [22]. Nevertheless, by re-assigning the connections, we can save one WDM while satisfying the capacities as shown in Figure 5.6(b). To reach this objective, we further present a min-cost max-flow network to re-allocate the connections. Due to its unimodular property, the assignment solution can be acquired directly without any approximation or rounding methodologies.

Figure 5.7 illustrates how the network flow model resolves the assignment problem. Since the horizontal and vertical connections are re-assigned independently with the same method, we just list the horizontal connections. Given the sets of connections and WDMs, a directed graph G is constructed as follows. There are four types of vertices contained in G: V_s and V_t represent the pseudo starting and ending nodes, respectively; V_C and V_W represent the connections of hyper nets and the WDMs which have already been placed. Also, there are three types of edges in G: $\{V_s \rightarrow V_C\}$, $\{V_C \rightarrow V_W\}$, and $\{V_W \rightarrow V_t\}$. The edges of the first type ensure that all the connections should be assigned to the WDMs; the edges of the second type determine which WDM will be chosen for assignment; and the edges of the third type guarantee that the WDM capacity constraint should be satisfied. Notably, in order not to disturb the routing results obtained in Section 5.3, we only allow each connection to connect with its neighboring WDMs, and the distance should be within dis_u, as shown in Figure 5.7. Meanwhile, the costs of edges are defined as follows: the cost from V_s to V_C is set to 0; the cost from V_C to V_W is the perpendicular distance between the WDM and the connection's current location; and the cost from V_W to V_t is the WDM usage cost. The capacities of edges are also defined: the capacities from V_W to V_t are the maximum allowable capacity, and the capacities of other edges are the number of passing nets through the connection for flow accommodation. Since our target is to reduce the costs of WDMs, we prefer to apply higher costs to the edges from V_W to V_t. Thus, we normalize the costs of edges from V_C to V_W so that the WDMs' usages are emphasized. After solving the example in Figure 5.6, the edges with flows are shown as solid lines in Figure 5.7. With the network model, three connections share two WDMs. It is seen that both the number of flow edges and vertices are proportional to the number of connections $|C|$, and the runtime complexity is within $\mathcal{O}(|C|^2)$.

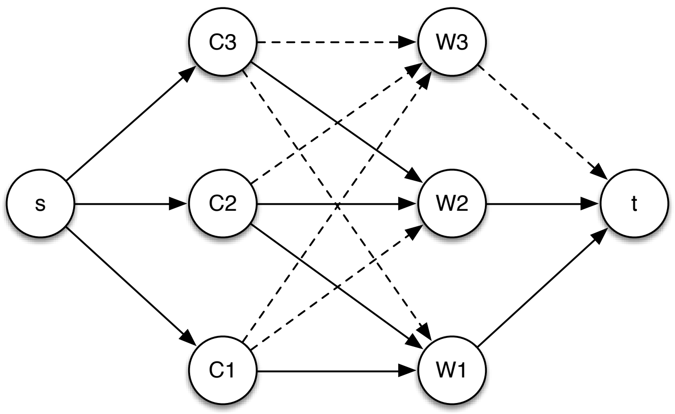


Figure 5.7: Example of min-cost max-flow assignment.

Table 5.1: Performance comparisons among different designs.

				Electrical [47]	Optical [22]	OPERON (ILP)		OPERON (LR)	
Bench	#Net	#HNet	#HPin	Power	Power	Power	CPU(s)	Power	CPU(s)
I1	2660	356	1306	20.50	4.92	4.79	> 3000	4.88	2.1
I2	1782	837	1701	50.79	14.48	12.39	> 3000	12.77	5.0
I3	5072	168	336	17.96	2.70	2.49	4.4	2.57	0.9
I4	3224	403	1474	21.51	5.70	5.45	341.3	5.62	2.4
I5	1994	933	1897	54.21	18.40	14.61	> 3000	15.22	8.8
average	-	-	-	32.99	9.24	7.95	> 1869.1	8.21	3.8
ratio	-	-	-	**3.565**	**1.000**	**0.860**	–	**0.889**	–

5.5 Experimental Results

We implemented the OPERON framework in C++, and tested it on a Linux machine with eight 3.3GHz CPUs. Meanwhile, we selected GUROBI [26] as our ILP solver, and open source graph library LEMON [2] as our min-cost max-flow network solver. For the optical parameters, we set the values of α, β to 1.5 dB/cm, 0.52 dB, with the same optical settings in [9]; and the power consumptions of modulators and detectors are set to 0.511 pJ/bit and 0.374 pJ/bit [77]. With these parameters, we derive the test cases from the industrial benchmarks, by up-scaling the dimension into centimeter scale, and employ the signal processing for the hyper net generation. The details of each test case are listed in the left part of Table 5.1. The column "**#Net**" gives the overall number of signal bits, while the column "**#HNet**" and "**#HPin**" correspond to the number of hyper nets and hyper pins, respectively. Since our work targets at optical-electrical co-design, we focus on the power consumption caused by both optical and electrical routes, as listed in column "**Power**". The optical power is calculated in Eq. (5.1), and the electrical power is estimated based on its dynamic power:

$$p_e = \gamma \cdot f \cdot V^2 \cdot Cap, \tag{5.6}$$

where γ, f, V, Cap denote the switching factor, system frequency, voltage level, and wire capacitance in proportional to the wire-length. Due to the signals' performance-critical nature, the wire-lengths of electrical wires are estimated based on the Rectilinear Steiner Minimum Tree (RSMT) with the parameters in [22, 77].

To show the effectiveness, we implemented the similar GLOW framework for optical designs [22], and the electrical design based on Streak [47] for comparison. From the experimental results, it is shown that the utilization of optical interconnects consumes the power costs about one-third of the counterpart caused by electrical wires. This proves the high efficiency of optical propagation for distant communications, consistent with the optical inherent characteristics. Nevertheless, in order to satisfy the loss constraint, a few hy-

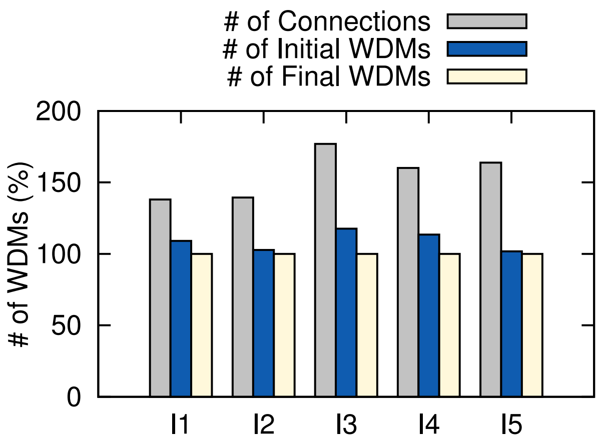


Figure 5.8: Comparison of WDMs for optical connections before the placement, before the assignment and after the assignment.

per nets cannot be routed on the optical layer and the residual nets have to be completed through electrical wires, resulting in additional power consumptions. To deal with this condition, we observe that after the employment of the optical-electrical paradigm, the overall power overheads are reduced by 14.0%. This is mainly because of the decreasing number of electrical routes, and the adjustment of EO/OE conversion also helps. However, due to the complexity of dealing with ILP formulation, the runtime overhead is significant, especially for the large benchmarks. Thus, the effectiveness of the proposed speed-up algorithm is shown, with the slightly worse performance but much shorter runtime.

To demonstrate the effectiveness of the WDM assignment, we compare the number of connections before the placement, and the WDMs before and after the assignment through the normalization in Figure 5.8. It is shown that the placement is able to reduce the usage of WDMs greatly for all the

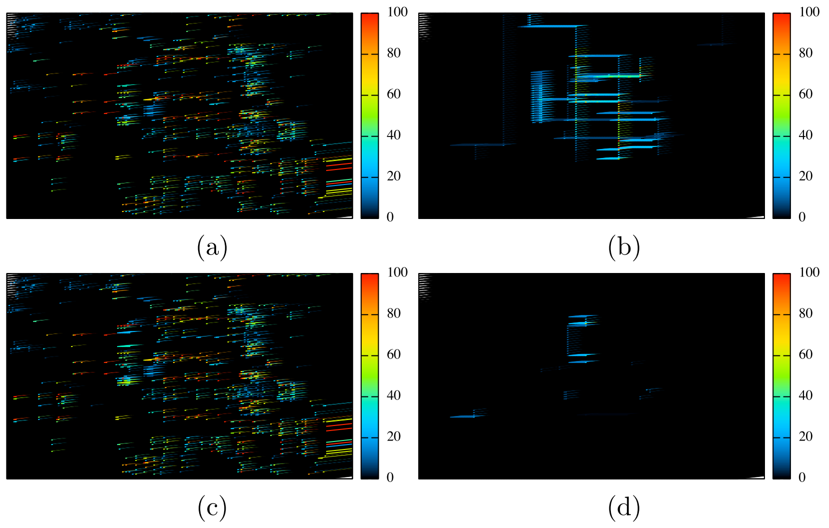


Figure 5.9: Power consumption distribution of I2. (a) Optical power in GLOW; (b) Electrical power in GLOW; (c) Optical power in OPERON; (d) Electrical power in OPERON.

cases, but still, suffers from the sub-optimality due to its heuristic style. With the integration of the network flow algorithm, we observe that the number of WDMs can be further reduced by 8.9% on average.

Finally, we perform the experiments to measure the normalized power hotspots on both optical and electrical layers for GLOW and OPERON, as shown in Figure 5.9. Figure 5.9(a) and Figure 5.9(b) provide the power distributions of GLOW, while Figure 5.9(c) and Figure 5.9(d) provide the distributions of OPERON. We observe that the hotspots in Figure 5.9(a) and Figure 5.9(c) are distributed in a very similar manner. It is reasonable because they employ the similar amounts of EO/OE conversion overheads. By comparison, from the electrical layers shown in Figure 5.9(b) and Figure 5.9(d), the hotspots are alleviated greatly in OPERON. Considering that a higher flexibility is allowed through optical-electrical co-design procedure, much fewer electrical wires are allocated on the electrical layer in Figure 5.9(d), consistent with our motivation. Therefore, the hotspot comparison demonstrates the power efficiency of the proposed OPERON framework.

5.6 Summary

In this chapter, a set of algorithms have been proposed for optical-electrical co-design of on-chip signals to optimize power consumptions. First, the clustering strategy generates the sets of hyper nets and hyper pins, for which the baseline topologies are constructed. Based on the baseline, the co-design solution set is developed for the minimization of total loss and power costs. Then a mathematical formulation targets at power optimization while satisfying the detection constraints, and follows a fast flow to reach a close performance with the ILP solution. Finally, a network flow model is adopted to utilize WDMs sufficiently. The results show that the route synthesis engine is able to provide the optical-electrical solution with legality and power efficiency. With the development of technologies, this work can open up opportunities for optical-electrical co-design research.

Chapter 6

Conclusion and Future Work

This dissertation proposes a set of novel algorithms for layer assignment and routing optimization. A timing-driven incremental layer assignment with slew optimization framework is proposed to handle delay optimization for all nets simultaneously while mitigating slew violations for saving buffering resources. The effectiveness has been demonstrated through both academia and industrial benchmarks. Furthermore, with critical path timing into consideration, another incremental layer assignment engine is presented with a more accurate timing formulation. The results show that our framework can achieve better results compared to the previously promoted layer assignment tool. Currently, for on-chip performance-critical signal groups, a synergistic topology generation and routing flow is devised with the efficient control of design regularity, besides the optimization of routability and wire-length. The industrial benchmarks demonstrate the effectiveness of the presented routing synthesis engine, where much fewer hotspots are shown on the congestion map compared to the manual designs from experienced designers. Considering the unique properties brought by optical interconnections, this dissertation also designs a novel optical-electrical co-design routing engine for on-chip signals to optimize the power consumption. The results show that it is able to provide more feasibilities for the optical-electrical solutions with legality, resulting in better power efficiency than the previous optical design.

After the discussion about the above methodologies for layering and routing optimization, it is explicitly shown that efficient routing can benefit the overall timing with legality. Meanwhile, since interconnect delay is becoming a bottleneck compared to cell delays with more advanced technology nodes, more efforts are required and should be dedicated to exploring the efficient interconnect paradigms. With these into consideration, there are some potentially interesting research topics to delve in:

- Concurrent delay and slew optimization for timing paths. Delay and slew are both important metrics for achieving timing closure, and also easily affected by layering and buffering optimization. For existing timing paths consisting of multiple nets and cells, appropriate routing and layering optimization is essential to satisfy the stringent timing requirement. Additionally, due to the important role of buffering, an advanced layer assignment should be integrated with buffering for timing optimization. Therefore, a coherent layer assignment and buffering tool is required to handle both delay and slew optimization simultaneously.

- Detailed exploration of signal routing. Since signal groups are required to share common topologies with parallel tracks, this may block available routing regions for pin accessibilities and lead to design rule violations in detailed routing. Hence a complete automatic flow from global to detailed routing is essential to utilize the resources intelligently and efficiently. Besides, more flexibilities can be introduced to construct the routing topologies for those signal bits, to make the balance between wire-length and path length. This can help to broaden the solution space and provide more competitive options for topology selection.

- Reliable optical-electrical co-design. To incorporate electrical wires with optical connections seamlessly, device-level reliability should be handled with great attention due to the sensitivity to environmental issues, such as thermal variation and so on. Thus appropriate mechanisms are necessitated to guarantee the functional correctness. Additionally, a more detailed optical model is required to incorporate more loss sources, such as coupling loss, modulation loss, etc. Similarly, the electrical power overheads should be calculated in a more accurate manner. By taking all of these factors into consideration, a comprehensive study should be given to optimize the resulting power while satisfying detection constraints.

Bibliography

[1] International roadmap for semiconductors. ITRS press conference, 2004.

[2] LEMON: library for efficient modeling and optimization in networks. `http://lemon.cs.elte.hu/trac/lemon`.

[3] OpenMP: An Industry-Standard API for Shared-Memory Programming. `http://www.openmp.org/`.

[4] Synopsys IC Compiler. `http://www.synopsys.com`.

[5] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications.* Prentice Hall/Pearson, 2005.

[6] Christoph Albrecht. Efficient incremental clock latency scheduling for large circuits. In *Proc. DATE*, pages 1–6, 2006.

[7] Jianchang Ao, Sheqin Dong, Song Chen, and Satoshi Goto. Delay-driven layer assignment in global routing under multi-tier interconnect structure. In *Proc. ISPD*, pages 101–107, 2013.

[8] Wim Bogaerts, Roel Baets, Pieter Dumon, Vincent Wiaux, Stephan Beckx, Dirk Taillaert, Bert Luyssaert, Joris Van Campenhout, Peter Bienstman, and Dries Van Thourhout. Nanophotonic waveguides in silicon-

on-insulator fabricated with CMOS technology. *Journal of Lightwave Technology*, 2005.

[9] Anja Boos, Luca Ramini, Ulf Schlichtmann, and Davide Bertozzi. Proton: An automatic place-and-route tool for optical networks-on-chip. In *Proc. ICCAD*, pages 138–145, 2013.

[10] Brian Borchers. CSDP, a C library for semidefinite programming. *Optimization Methods and Software*, 11:613–623, 1999.

[11] Rohit Chandra. *Parallel Programming in OpenMP*. Morgan Kaufmann, 2001.

[12] Hua-Yu Chang, IH-R Jiang, and Yao-Wen Chang. Timing ECO optimization via Bézier curve smoothing and fixability identification. *IEEE TCAD*, 31(12):1857–1866, 2012.

[13] Yen-Jung Chang, Yu-Ting Lee, Jhih-Rong Gao, Pei-Ci Wu, and Ting-Chi Wang. NTHU-Route 2.0: a robust global router for modern designs. *IEEE TCAD*, 29(12):1931–1944, 2010.

[14] James Hsueh-Chung Chen, Theodorus E Standaert, Emre Alptekin, Terry A Spooner, and Vamsi Paruchuri. Interconnect performance and scaling strategy at 7 nm node. In *Proc. IITC*, pages 93–96, 2014.

[15] Minsik Cho and David Z Pan. BoxRouter: a new global router based on box expansion and progressive ILP. *IEEE TCAD*, 26(12):2130–2143, 2007.

[16] Chris Chu and Yiu-Chung Wong. FLUTE: Fast lookup table based rectilinear steiner minimal tree algorithm for VLSI design. *IEEE TCAD*, 27(1):70–83, 2008.

[17] Jason Cong. An interconnect-centric design flow for nanometer technologies. *Proceedings of the IEEE*, 89(4):505–528, 2001.

[18] Jason Cong and Bin Liu. A metric for layout-friendly microarchitecture optimization in high-level synthesis. In *Proc. DAC*, pages 1239–1244, 2012.

[19] Olivier Coudert, Jason Cong, Sharad Malik, and Majid Sarrafzadeh. Incremental CAD. In *Proc. ICCAD*, pages 236–244, 2000.

[20] Ke-Ren Dai, Wen-Hao Liu, and Yih-Lang Li. Efficient simulated evolution based rerouting and congestion-relaxed layer assignment on 3-D global routing. In *Proc. ASPDAC*, pages 570–575, 2009.

[21] Ke-Ren Dai, Wen-Hao Liu, and Yih-Lang Li. NCTU-GR: efficient simulated evolution-based rerouting and congestion-relaxed layer assignment on 3-D global routing. *IEEE VLSI*, 20(3):459–472, 2012.

[22] Duo Ding, Bei Yu, and David Z Pan. GLOW: A global router for low-power thermal-reliable interconnect synthesis using photonic wavelength multiplexing. In *Proc. ASPDAC*, pages 621–626, 2012.

[23] Duo Ding, Yilin Zhang, Haiyu Huang, Ray T Chen, and David Z Pan. O-router: an optical routing framework for low power on-chip silicon nano-photonic integration. In *Proc. DAC*, pages 264–269, 2009.

[24] Peng Du, Shih-Hung Weng, Xiang Hu, and Chung-Kuan Cheng. Power grid sizing via convex programming. In *Proc. ASICON*, pages 337–340, 2011.

[25] Ananda D Gunawardena, SK Jain, and Larry Snyder. Modified iterative methods for consistent linear systems. *Linear Algebra and its Applications*, 154:123–143, 1991.

[26] Gurobi Optimization Inc. Gurobi optimizer reference manual. `http://www.gurobi.com`, 2016.

[27] Ou He, Sheqin Dong, Jinian Bian, Sotoshi Goto, and Chung-Kuan Cheng. Bus via reduction based on floorplan revising. In *Proc. GLSVLSI*, pages 9–14, 2010.

[28] Chin-Hsiung Hsu, Huang-Yu Chen, and Yao-Wen Chang. Multi-layer global routing considering via and wire capacities. In *Proc. ICCAD*, pages 350–355, 2008.

[29] Chin-Hsiung Hsu, Huang-Yu Chen, and Yao-Wen Chang. Multilayer global routing with via and wire capacity considerations. *IEEE TCAD*, 29(5):685–696, 2010.

[30] Meng-Kai Hsu, Nitesh Katta, Homer Yen-Hung Lin, Keny Tzu-Hen Lin, King Ho Tam, and Ken Chung-Hsing Wang. Design and manufacturing process co-optimization in nano-technology. In *Proc. ICCAD*, pages 574–581, 2014.

[31] Shiyan Hu, Charles J Alpert, Jiang Hu, Shrirang K Karandikar, Zhuo Li, Weiping Shi, and Chin Ngai Sze. Fast algorithms for slew-constrained minimum cost buffering. *IEEE TCAD*, 26(11):2009–2022, 2007.

[32] Shiyan Hu, Zhuo Li, and Charles J Alpert. A polynomial time approximation scheme for timing constrained minimum cost layer assignment. In *Proc. ICCAD*, pages 112–115, 2008.

[33] Shiyan Hu, Zhuo Li, and Charles J Alpert. A faster approximation scheme for timing driven minimum cost layer assignment. In *Proc. ISPD*, pages 167–174, 2009.

[34] Andrew B. Kahng, Bao Liu, and Sheldon X-D. Tan. Efficient decoupling capacitor planning via convex programming methods. In *Proc. ISPD*, pages 102–107, 2006.

[35] Andrew B. Kahng and Gabriel Robins. A new class of iterative steiner tree heuristics with good performance. *IEEE TCAD*, 11(7):893–902, 1992.

[36] Shrirang K Karandikar, Charles J Alpert, Mehmet Can Yildiz, Paul Villarrubia, Steve Quay, and Tuhin Mahmud. Fast electrical correction

using resizing and buffering. In *Proc. ASPDAC*, pages 553–558, 2007.

[37] Chandramouli V Kashyap, Charles J Alpert, Frank Liu, and Anirudh Devgan. Closed form expressions for extending step delay and slew metrics to ramp inputs. In *Proc. ISPD*, pages 24–31, 2003.

[38] Dae Hyun Kim and Sung Kyu Lim. Bus-aware microarchitectural floorplanning. In *Proc. ASPDAC*, pages 204–208, 2008.

[39] Dae Hyun Kim and Sung Kyu Lim. Global bus route optimization with application to microarchitectural design exploration. In *Proc. ICCD*, pages 658–663, 2008.

[40] Yukihide Kohira, Tomomi Matsui, Yoko Yokoyama, Chikaaki Kodama, Atsushi Takahashi, Shigeki Nojima, and Satoshi Tanaka. Fast mask assignment using positive semidefinite relaxation in LELECUT triple patterning lithography. In *Proc. ASPDAC*, pages 665–670, 2015.

[41] Tushar Krishna, Arya Balachandran, Siau Ben Chiah, Li Zhang, Bing Wang, Cong Wang, Kenneth Lee Eng Kian, Jurgen Michel, and Li-Shiuan Peh. Automatic place-and-route of emerging LED-driven wires within a monolithically-integrated cmos-III-V process. In *Proc. DATE*, pages 344–349, 2017.

[42] Jill H. Y. Law and Evangeline F. Y. Young. Multi-bend bus driven floorplanning. In *Proc. ISPD*, pages 113–120, 2005.

[43] Tsung-Hsien Lee, Yen-Jung Chang, and Ting-Chi Wang. An enhanced global router with consideration of general layer directives. In *Proc. ISPD*, pages 53–60, 2011.

[44] Tsung-Hsien Lee and Ting-Chi Wang. Congestion-constrained layer assignment for via minimization in global routing. *IEEE TCAD*, 27(9):1643–1656, 2008.

[45] Tsung-Hsien Lee and Ting-Chi Wang. Simultaneous antenna avoidance and via optimization in layer assignment of multi-layer global routing. In *Proc. ICCAD*, pages 312–318, 2010.

[46] Zhuo Li, Charles J Alpert, Shiyan Hu, Tuhin Muhmud, Stephen T Quay, and Paul G Villarrubia. Fast interconnect synthesis with layer assignment. In *Proc. ISPD*, pages 71–77, 2008.

[47] Derong Liu, Vinicius Livramento, Salim Chowdhury, Duo Ding, Huy Vo, Akshay Sharma, and David Z Pan. Streak: synergistic topology generation and route synthesis for on-chip performance-critical signal groups. In *Proc. DAC*, pages 1–6, 2017.

[48] Derong Liu, Bei Yu, Salim Chowdhury, and David Z Pan. Incremental layer assignment for critical path timing. In *Proc. DAC*, 2016.

[49] Derong Liu, Bei Yu, Salim Chowdhury, and David Z Pan. Incremental layer assignment for timing optimization. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 22(4):75, 2017.

[50] Derong Liu, Bei Yu, Salim Chowdhury, and David Z. Pan. TILA-S: Timing-driven incremental layer assignment avoiding slew violations. *IEEE TCAD*, 37(1):231–244, 2018.

[51] Derong Liu, Bei Yu, Vinicius Livramento, Salim Chowdhury, Duo Ding, Huy Vo, Akshay Sharma, and David Z Pan. Synergistic topology generation and route synthesis for on-chip performance-critical signal groups. *IEEE TCAD*, 2018.

[52] Derong Liu, Zheng Zhao, Zheng Wang, Zhoufeng Ying, Ray T Chen, and David Z Pan. OPERON: optical-electrical power-efficient route synthesis for on-chip signals. In *Proc. DAC*, page 75, 2018.

[53] Wen-Hao Liu, Wei-Chun Kao, Yih-Lang Li, and Kai-Yuan Chao. NCTU-GR 2.0: multithreaded collision-aware global routing with bounded-length maze routing. *IEEE TCAD*, 32(5):709–722, 2013.

[54] Wen-Hao Liu and Yih-Lang Li. Negotiation-based layer assignment for via count and via overflow minimization. In *Proc. ASPDAC*, pages 539–544, 2011.

[55] Vinicius Livramento, Derong Liu, Salim Chowdhury, Bei Yu, Xiaoqing Xu, David Z Pan, Jose Luis Guntzel, and Luiz CV dos Santos. Incremental layer assignment driven by an external signoff timing engine. *IEEE TCAD*, 36(7):1126–1139, 2017.

[56] Stuart Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory (TIT)*, 1982.

[57] Tao Luo, David A. Papa, Zhuo Li, C. N. Sze, Charles J. Alpert, and David Z. Pan. Pyramids: an efficient computational geometry-based approach for timing-driven placement. In *Proc. ICCAD*, pages 204–211, 2008.

[58] Tilen Ma and Evangeline F. Y. Young. TCG-based multi-bend bus driven floorplanning. In *Proc. ASPDAC*, pages 192–197, 2008.

[59] R Garey Michael and S Johnson David. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., 1979.

[60] Fan Mo and Robert K Brayton. Semi-detailed bus routing with variation reduction. In *Proc. ISPD*, pages 143–150, 2007.

[61] Fan Mo and Robert K. Brayton. A simultaneous bus orientation and buses pin flipping algorithm. In *Proc. ICCAD*, pages 386–389, 2007.

[62] Michael D Moffitt. MaizeRouter: Engineering an effective global router. *IEEE TCAD*, 27(11):2017–2026, 2008.

[63] Moustafa Mohamed, Zheng Li, Xi Chen, Li Shang, Alan Mickelson, Manish Vachharajani, and Yihe Sun. Power-efficient variation-aware photonic on-chip network management. In *Proc. ISLPED*, 2010.

[64] Nicholas J. Naclerio, Sumio Masuda, and Kazuo Nakajima. The via minimization problem is NP-complete. *IEEE Transactions on Computers*, 38(11):1604–1608, 1989.

[65] Gi-Joon Nam, Cliff Sze, and Mehmet Yildiz. The ISPD global routing benchmark suite. In *Proc. ISPD*, pages 156–159, 2008.

[66] Sudeep Pasricha, Nikil Dutt, Elaheh Bozorgzadeh, and Mohamed Ben-Romdhane. Floorplan-aware automated synthesis of bus-based communication architectures. In *Proc. DAC*, pages 565–570, 2005.

[67] Sudeep Pasricha, Nikil D Dutt, Elaheh Bozorgzadeh, and Mohamed Ben-Romdhane. Fabsyn: Floorplan-aware bus architecture synthesis. *IEEE VLSI*, 14(3):241–253, 2006.

[68] Andrea Peano, Luca Ramini, Marco Gavanelli, Maddalena Nonato, and Davide Bertozzi. Design technology for fault-free and maximally-parallel wavelength-routed optical networks-on-chip. In *Proc. ICCAD*, page 3, 2016.

[69] Yuantao Peng and Xun Liu. Low-power repeater insertion with both delay and slew rate constraints. In *Proc. DAC*, pages 302–307, 2006.

[70] G. Persky and L. V. Tran. Topological routing of multi-bit data buses. In *Proc. DAC*, pages 679–682, 1984.

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

[72] JA Roy and IL Markov. High-performance routing at the nanometer scale. *IEEE TCAD*, 27(6):1066–1077, 2008.

[73] Jarrod A Roy and Igor L Markov. ECO-system: Embracing the change in placement. *IEEE TCAD*, 26(12):2173–2185, 2007.

[74] Subhendu Roy, Pavlos M Mattheakis, Laurent Masse-Navette, and David Z Pan. Clock tree resynthesis for multi-corner multi-mode timing closure. In *Proc. ISPD*, pages 69–76, 2014.

[75] Andrzej P Ruszczyński. *Nonlinear Optimization*, volume 13. Princeton university press, 2006.

[76] Daohang Shi, Edward Tashjian, and Azadeh Davoodi. Dynamic planning of local congestion from varying-size vias for global routing layer assignment. In *Proc. ASPDAC*, pages 372–377, 2016.

[77] Chen Sun, Mark Wade, Michael Georgas, Sen Lin, Luca Alloatti, Benjamin Moss, Rajesh Kumar, Amir Atabaki, Fabio Pavanello, Rajeev Ram, et al. A 45nm SOI monolithic photonics chip-to-chip link with bit-statistics-based resonant microring thermal tuning. In *Proc. VLSIC*, pages C122–C123, 2015.

[78] Chen Sun, Mart T Wade, Yunsup Lee, Jason S Orcutt, Luca Alloatti, Michael S Georgas, Andrew S Waterman, Jeffrey M Shainline, Rimas R Avizienis, Sen Lin, et al. Single-chip microprocessor that communicates directly using light. *Nature*, 528(7583):534, 2015.

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

[80] Lieven Vandenberghe, Stephen Boyd, and Abbas El Gamal. Optimal wire and transistor sizing for circuits with non-tree topology. In *Proc. ICCAD*, pages 252–259, 1997.

[81] Pei-Ci Wu, Qiang Ma, and Martin D. F. Wong. An ILP-based automatic bus planner for dense PCBs. In *Proc. ASPDAC*, pages 181–186, 2013.

[82] Po-Hsun Wu and Tsung-Yi Ho. Bus-driven floorplanning with bus pin assignment and deviation minimization. *Integration, the VLSI Journal*, 45(4):405–426, 2012.

[83] Rui Wu, Chin-Hui Chen, Cheng Li, Tsung-Ching Huang, Fan Lan, Chong Zhang, Yun Pan, John E Bowers, Raymond G Beausoleil, and Kwang-Ting Cheng. Variation-aware adaptive tuning for nanophotonic interconnects. In *Proc. ICCAD*, pages 487–493, 2015.

[84] Tai-Hsuan Wu, Azadeh Davoodi, and Jeffrey T Linderoth. GRIP: Global routing via integer programming. *IEEE TCAD*, 30(1):72–84, 2011.

[85] Hua Xiang, Liang Deng, Li-Da Huang, and Martin DF Wong. Opc-friendly bus driven floorplanning. In *Proc. ISQED*, pages 847–852, 2007.

[86] Hua Xiang, Xiaoping Tang, and Martin D. F. Wong. Bus-driven floorplanning. *IEEE TCAD*, 23(11):1522–1530, november 2004.

[87] Jin-Tai Yan. Efficient layer assignment of bus-oriented nets in high-speed PCB designs. *IEEE TCAD*, 35(8):1332–1344, 2016.

[88] Tan Yan and Martin D. F. Wong. Untangling twisted nets for bus routing. In *Proc. ICCAD*, pages 396–400, 2007.

[89] Tan Yan and Martin DF Wong. BSG-Route: A length-matching router for general topology. In *Proc. ICCAD*, pages 499–505, 2008.

[90] Yunfeng Yang, Wai-Shing Luk, David Z. Pan, Hai Zhou, Changhao Yan, Dian Zhou, and Xuan Zeng. Layout decomposition co-optimization for hybrid e-beam and multiple patterning lithography. *IEEE TCAD*, 35(9):1532–1545, 2016.

[91] Bei Yu, Derong Liu, Salim Chowdhury, and David Z. Pan. TILA: Timing-driven incremental layer assignment. In *Proc. ICCAD*, pages 110–117, 2015.

[92] Bei Yu, Kun Yuan, Duo Ding, and David Z. Pan. Layout decomposition for triple patterning lithography. *IEEE TCAD*, 34(3):433–446, March 2015.

[93] Yanheng Zhang and Chris Chu. GDRouter: Interleaved global routing and detailed routing for ultimate routability. In *Proc. DAC*, pages 597–602, 2012.

[94] Yilin Zhang, Ashutosh Chakraborty, Salim Chowdhury, and David Z Pan. Reclaiming over-the-IP-block routing resources with buffering-aware rectilinear steiner minimum tree construction. In *Proc. ICCAD*, pages 137–143, 2012.

[95] Zheng Zhao, Zheng Wang, Zhoufeng Ying, Shounak Dhar, Ray T. Chen, and David Z. Pan. Logic synthesis for energy-efficient photonic integrated circuits. In *Proc. ASPDAC*, pages 355–360, 2018.

Vita

Derong Liu received the B.S. degree in microelectronics from Fudan University, Shanghai, China, in 2011, and the M.S. degree in engineering from the University of Texas at Austin, Texas, US, in 2016. She started her Ph.D. program at the University of Texas at Austin in 2014, with the supervision of Prof. David Z. Pan. She has interned at Oracle, Santa Clara in 2016 summer, and Cadence, Austin in 2015 summer, 2017 spring and summer. Derong Liu's research interests include physical design and logic synthesis for digital and optical technologies.

Permanent address: derongliuliu@gmail.com

This dissertation was typeset with LaTeX[†] by the author.

[†]LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.