

Incremental Layer Assignment for Critical Path Timing

Derong Liu¹, Bei Yu², Salim Chowdhury³, and David Z. Pan¹

¹ECE Department, University of Texas at Austin, Austin, TX, USA

²CSE Department, Chinese University of Hong Kong, NT, Hong Kong

³Oracle Corp., Austin, TX, USA

{deronliu,dpan}@cerc.utexas.edu, byu@cse.cuhk.edu.hk, salim.chowdhury@oracle.com

ABSTRACT

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

1. INTRODUCTION

In emerging technology nodes, transistor and interconnect feature sizes are further scaling into nanometer regime, thus timing issues on interconnect lines are prevalent in modern design closure [1]. As an integral part of the timing convergence flow, global routing determines the topologies of all nets, and thus is critical for performance optimization [2].

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 critical paths, layer assignment should also pay attention to the critical paths to avoid potential timing violations. Besides, in advanced technology nodes, resistance and capacitance values vary significantly among different metal layers [3]: higher metal layers are wider with lower resistance, while lower metal layers are thinner with higher resistance values. Thus, high metal layers are more attractive for longer nets that may introduce serious timing issues. Nevertheless, since there exist edge capacity constraints for each metal layer, not all segments are allowed to be assigned on higher metal layers. The segments leading to critical sinks are preferred to be assigned on high layers to reduce the potential violations. Therefore, an intelligent layer assignment framework is necessary to reduce the critical path timing [4].

There are many layer assignment works, targeting at via number minimization, antenna effect avoidance, and timing optimization, etc [4–9]. For via minimization, a polynomial-time al-

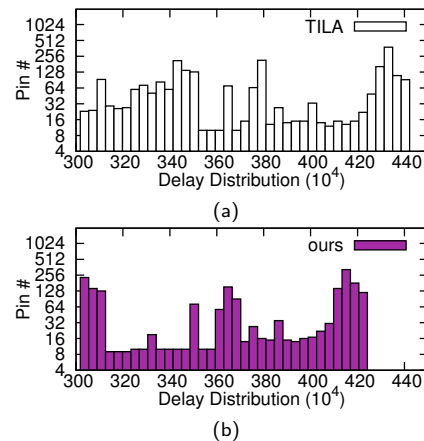


Figure 1: Pin delay distribution of critical nets for benchmark adaptec1, where 0.5% of the nets are released as critical nets. (a) Results from TILA [4]; (b) Results from our incremental layer assignment framework.

gorithm determines the net order and then solves one net each time through dynamic programming considering congestion issues [5]. Dai et al. [6] also apply a dynamic programming for net-by-net layer assignment. However, the fixed net order affects the final performance because nets with higher priorities have more layer selections while those nets with lower priorities only have limited resources. To alleviate the net order limitation, Liu et al. [7] adopt a negotiation-based methodology to minimize via count and capacity violations. Meanwhile, antenna avoidance is included during layer assignment where via counts are also reduced through min-cost max-flow model [8]. Ao et al. [9] focus on optimizing via counts and net delay. Nevertheless, the via capacity model is not considered thus more wires may be assigned on high metal layers, resulting in illegal solutions. Very recently, Yu et al. [4] propose an incremental layer assignment integrated with timing optimization engine. The proposed framework, TILA, is able to provide a global view of minimizing the total net delay for the selected nets.

Although TILA [4] 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 timing model is based on weighted sum delay of all the segments. Therefore, although TILA can effectively improve the total net delay, for some critical nets, the worst path delay may still violate timing budget. (2) The optimization engine of TILA is based on Lagrangian relaxation, whose performance may heavily rely on the initial values of multipliers. (3) In addition, when via delay and via capacity are considered, layer assignment is similar to quadratic assignment problem [10], which is essentially a non-linear optimization problem. However, to achieve extremely fast speed, TILA artificially approximates some quadratic terms to

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](http://permissions.acm.org).

DAC '16, June 05-09, 2016, Austin, TX, USA

© 2016 ACM. ISBN 978-1-4503-4236-0/16/06...\$15.00

DOI: <http://dx.doi.org/10.1145/2897937.2898033>

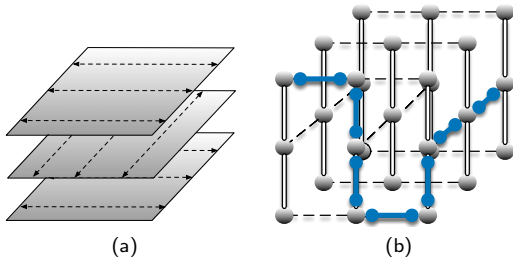


Figure 2: Layer and grid model. (a) Layer model; (b) Net routing on grid model.

linear model, which may impact the layer assignment accuracy and performance.

In this paper, we propose a novel incremental layer assignment framework targeting at critical path timing optimization. Our layer assignment tool can overcome all three limitations of TILA, thus is able to achieve better timing performance. Fig. 1 compares the layer assignment results between TILA and our work. Fig. 1(a) gives the results from TILA, where more pins have delay over 4.2×10^6 . On the other hand, from Fig. 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 contributions of our work are listed as follows.

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

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

2. PRELIMINARIES

2.1 Graph Model

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

For x/y -direction edges, each of them has a specified routing capacity on different layers, 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 $cap_e(l)$. Similarly, there is also a certain via capacity constraint for vias passing each tile. The via capacity constraint is determined by the available routing capacity of these two edges connecting this vertex, and is computed as follows [11].

$$cap_g(l) = \lfloor \frac{(w_w + w_s) \cdot Tile_w \cdot (cap_{e_0}(l) + cap_{e_1}(l))}{(v_w + v_s)^2} \rfloor, \quad (1)$$

where $w_w, w_s, v_w, v_s, Tile_w$ represent wire width, wire spacing, via width, via spacing and tile width, respectively. For vias between two layers, each layer have two edges connecting with grid g , i.e. e_0 and e_1 , whose routing capacity are represented by $cap_{e_0}(l), cap_{e_1}(l)$. From Eqn. (1), we can see that if these two connected edges are full of routing wires, then no vias are allowed to pass through this grid.

2.2 Timing Model

To calculate 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. Eqn. (2) gives the timing cost calculation of segment s_i on layer j .

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

where $R(l), C_e(l)$ refer to the wire resistance and capacitance of the layer l on which segment i is assigned, and $C_d(i)$ is the downstream capacitance of segment i . During calculating $C_d(i)$, the layer assignment of all the segments driven by segment i should be considered. Thus, we compute $C_d(i)$ from sinks to source in a bottom-to-up manner. Similarly, via timing cost is calculated as in Eqn. (3), which is determined by via resistance and the minimum downstream capacitance of its connected segments [4].

$$t_v(i, j, p, q) = \sum_{l=j}^{q-1} R_v(l) \cdot \min\{C_d(i), C_d(p)\}, \quad (3)$$

where $R_v(l)$ is the resistance of via between layers l and $l+1$, and we assume layer j is lower than layer q .

2.3 Problem Formulation

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

Problem 1 (CPLA). *Given a 3-D grid graph, edge and layer information, initial routing and layer assignment, and set of critical nets, layer assignment re-assigns layers among critical and non-critical nets onto layers in order to minimize their maximum path timing and satisfy the edge capacity constraints.*

3. CPLA ALGORITHMS

In this section, we discuss the details of our framework to solve the CPLA problem. First we propose an integer linear programming (ILP) formulation. Then we relax this formulation into 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. Finally, we give the mapping algorithm to locate integer solutions.

3.1 ILP Formulation

In this work, our objective is to optimize the critical path timing of selected critical nets, including both segment delays and via delays. As introduced in Section 2, segment delay can be calculated based on Eqn. (2), and via delay based on Eqn. (3). Then, we propose the integer linear programming (ILP) formulation as shown in formula (4). This formulation concerns all the segments and vias along the critical paths, and also those on the branches connected with critical paths since their assignments would affect the downstream capacitance driven by the critical paths. For convenience, notations used are listed in Table 1.

$$\begin{aligned}
\min \quad & \sum_{i \in S(Nc)} \sum_j^L t_s(i, j) \cdot x_{ij} + \\
& \sum_{i, p \in Sx(Nc)} \sum_j^L \sum_q^L t_v(i, j, p, q) \cdot y_{ijpq}, \quad (4a) \\
\text{s.t.} \quad & \sum_j x_{ij} = 1, \quad \forall i \in S(Nc) \quad j \in L, \quad (4b) \\
& \sum_{i \in S(e)} x_{ij} \leq \text{cap}_e(j), \quad \forall e \in E, \quad (4c) \\
& \sum_{(i, p) \in Sx(Nc)} y_{ijpq} + n_v(x_{ij} + x_{pq}) \leq \text{cap}_g(l), \quad \forall l, j < l < q, \quad (4d) \\
& x_{ij} \geq y_{ijpq}, \quad (i, p) \in Sx(Nc) \quad j, q \in L, \quad (4e) \\
& x_{pq} \geq y_{ijpq}, \quad (i, p) \in Sx(Nc) \quad j, q \in L, \quad (4f) \\
& x_{ij} + x_{pq} \leq y_{ijpq} + 1, \quad (i, p) \in Sx(Nc) \quad j, q \in L, \quad (4g) \\
& y_{ijpq} \text{ is binary}, \quad (i, p) \in Sx(Nc) \quad j, q \in L, \quad (4h) \\
& x_{ij} \text{ is binary}, \quad i \in S(Nc) \quad j \in L. \quad (4i)
\end{aligned}$$

In our mathematical formulation, constraint (4b) guarantees that one segment can be assigned on one and only one layer. Constraint (4c) sets the routing wire limit for edge e on layer j , i.e. $\text{cap}_e(j)$. Notably, while calculating $\text{cap}_e(l)$, we also consider the number of those non-released segments on this edge besides its initial routing capacity. Thus, for incremental assignment problem, the edge capacity constraint is more stringent than the initial problem. Similarly, constraint (4d) places the limitation of the via number to pass through each grid g for different layers, and those vias caused by those non-released segments should also be taken into accounts. Meanwhile, we should notice that via capacity is also affected by the segments assigned on neighboring edges, where n_v is to represent the via number on one routing track within one tile. Besides, constraints (4e)–(4g) represent that y_{ijpq} is the product of x_{ij} and x_{pq} , because all x_{ij} and y_{ijpq} are binaries according to constraint (4h) and (4i).

Nevertheless, there is a potential problem for constraint (4d). If via capacity violations already exist in initial layer assignment inputs and cannot be eliminated completely, this constraint may be so stringent that no legal solutions can be obtained. Therefore, we relax this constraint by adding one variable V_o , representing the allowed maximum violation number. Then constraint (4d) should be re-written as follows,

$$\sum_{(i, p) \in Sx(Nc)} y_{ijpq} + n_v \cdot (x_{ij} + x_{pq}) \leq \text{cap}_g(l) + V_o, \quad \forall l, j < l < p.$$

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. Similar to [4], 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 considerable run-time. In order to alleviate this overhead, some speed-up techniques are introduced in following sections.

3.2 Self-Adaptive Partition Algorithm

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 [12], the newly updated assignment results of neighboring partitions benefit each current partition. Fig. 3(a)

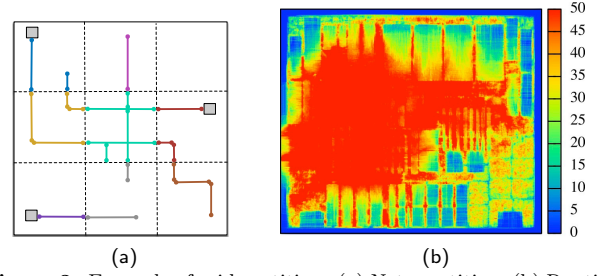


Figure 3: Example of grid partition. (a) Nets partition; (b) Routing density for benchmark adaptec1 by NCTU-GR.

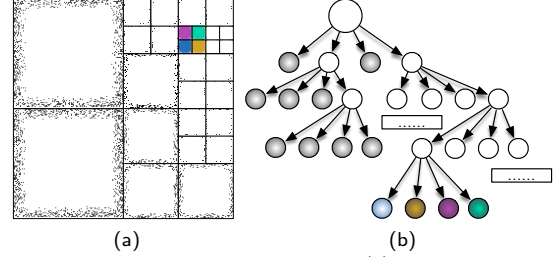


Figure 4: Sub-grid partition illustration. (a) Sub-grid partition; (b) Sub-grid corresponding partition tree.

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(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 calculation 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 similar number of critical segments.

Fig. 4(a) gives the example of partition results for the lower left one in 5×5 divisions, where each division contains similar number of critical segments. To achieve this, 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. 4(b) shows the quadruple tree corresponding to Fig. 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 further partition should not be allowed in fact. To avoid this condition, 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. 4(a). There are two leaf nodes in the first level representing these two left partitions. In Fig. 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 similar problem size, each thread deals with a workload in a well-balanced manner.

3.3 Semidefinite Programming Relaxation

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

Table 1: Notations used in this paper.

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
$S(Nc)$	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
S_e	set of segments on edge $e \in E$
$V(s_i, s_p)$	via connecting segment s_i and segment s_p
x_{ij}	binary variable, set to 1 if s_i is assigned to layer j
$t_s(i, j)$	timing cost when s_i is assigned to layer j
$t_v(i, j, p, q)$	timing cost of via v from layer j to $q - 1$ where $v = V(s_i, s_p)$
$cap_e(l)$	available routing capacity of edge e on layer l
$cap_g(l)$	available via capacity for node g on layer l

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 as Linear Programming (LP), but it is more general than LP due to its symmetric matrix forms. SDP is solvable in polynomial time, while it provides a theoretically better solution than LP [13]. 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:

$$\min (T \cdot X). \quad (5)$$

In Eqn. (5), matrix T and X are both $|S| \cdot |L|$ -dimension symmetric matrices, where $|S|$ is the number of critical segments in each partition and $|L|$ is the number of layers. Eqn. (6) shows all coefficients in matrix 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 into layer j and layer q , respectively. Each $t_v(i, j, p, q)$ is in the same row as $t_s(i, j)$ and the same column as $t_s(p, q)$. Matrix X in Eqn. (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} .

$$T = \begin{pmatrix} t_s(i, j) & \dots & t_v(i, j, p, q) \\ \dots & \dots & \dots \\ t_v(i, j, p, q) & \dots & t_s(p, q) \end{pmatrix}, \quad (6)$$

$$X = \begin{pmatrix} x_{ij} & \dots & y_{ijpq} \\ \dots & \dots & \dots \\ y_{ijpq} & \dots & x_{pq} \end{pmatrix}. \quad (7)$$

For each x_{ij} , it is expected to be binary and placed in the diagonal line of objective matrix 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 (4e)–(4g), which also apply for continuous solutions. Because constraints (4e)–(4g) 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 Eqn. (4a)).

To guarantee an effective solution, the constraints in ILP formulation (4) should also be included in SDP. Constrains (4b) and (4c) can be directly formulated into SDP since they are linear constraints. Different from (4b), (4c) requires slack variables in the objective matrix for 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 (4d), we prefer to move it into the objective matrix by adding the penalty to save the run-time. Then penalty is represented as $\lambda_{i,j,p,q}$, which is added to $t_v(i, j, p, q)$ in matrix

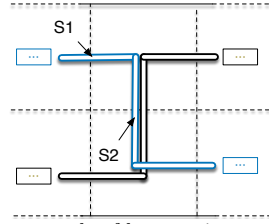


Figure 5: An example of layer assignment through SDP.

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

Figure 6: T matrix and solution X matrix of the example.

T . The penalty is calculated by dividing the existing via number by its capacity.

To make it more clear, here we give an example of how SDP can be applied to layer assignment problem. Fig. 5 shows a part of one net. Due to 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 matrix T and 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 T are basically x_{ij} s, representing whether they are assigned on the corresponding layers. The entries on the same column and row with x_{ij} and x_{pq} represent the potential via costs from layer j to layer q . Based on Fig. 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. 6 we list an example of matrix T , as well as the matrix X after solving the SDP. From matrix X , we can see that segment s_1 should be assigned on layer 3 as x_{12} is very close to 1. Meanwhile, for s_2 , its x_{ij} is 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 post mapping algorithm to determine on which layer it should be assigned.

3.4 Post 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 mapping algorithm to transfer a continuous SDP solution into a discrete layer assignment solution.

Algorithm 1 Post Mapping Algorithm

Input: Solution matrix 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**

The details of our mapping algorithm is shown in Alg. 1, whose input is the original solution matrix X . Initially, we read all the solution entries, and save those x_{ij} s to each correspond-

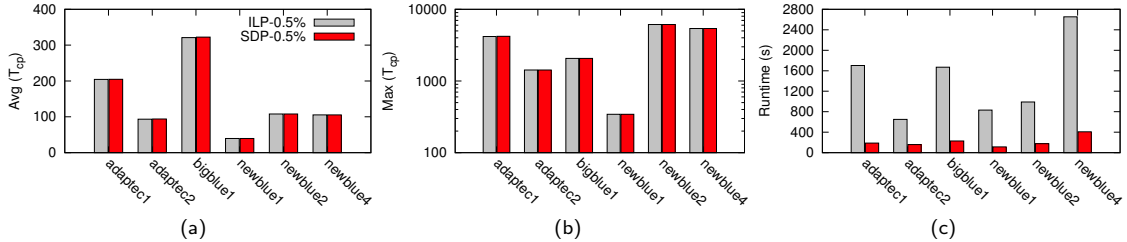


Figure 7: Comparison between ILP and SDP on some small test cases: (a) on average delay for all critical paths; (b) on maximum delay for all critical paths; (c) on runtime.

ing segment. Then we traverse each edge with critical 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 l of edge e , there is a specified edge capacity constraint, i.e. $cap_e(j)$. This means that the number of 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). 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 $O(|E||L|\log|S_e|)$, 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.

4. EXPERIMENTAL RESULTS

The proposed layer assignment framework is implemented in C++, and tested on a Linux machine with 2.9 GHz Intel(R) Core and 192 GB memory. We select GUROBI [14] as the ILP solver, and CSDP [15] as the SDP solver. Besides, we utilize OpenMP [16] for parallel computing. As that in [4], we test our framework on ISPD’08 global routing benchmarks [17]. It should be noted in our experiments, both the resistance and capacitance values are from industrial settings, thus our experimental results may have better agreement with industry timing.

In the first experiment, we compare the ILP formulation (see Section 3.1) with the SDP based methodology (see Section 3.3 and Section 3.4). Since ILP formulation may suffer from 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. 7. Note that partitioning technique is applied to both methods. We can see from Fig. 7(a) and Fig. 7(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. 7(c)).

In the second experiment, we further evaluate our SDP based method by comparing it with TILA [4]. To make a fair comparison, we release the same set of nets for both TILA and our SDP. Table 2 lists the comparison results for 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 “OV#” and “via#” list via capacity overflow and via count. The run-time is also reported in the Column “CPU(s)”. From Table 2 we can see that comparing with TILA, our SDP-based method can reduce the average timing by 14%, while the maximum timing can also be decreased by 4%. Since TILA also devotes efforts in maximum timing optimization, the improve-

ment of maximum timing is reasonable. Our work also reduces the via violation number by 10%, and keep the same via count number as TILA. In addition, the reported runtime of SDP increases by 3.16 times in comparison with TILA, due to the nature that SDP problem is more complicated than min-cost flow problem. However, since we propose adaptive partitioning in SDP based method (see Section 3.2), the SDP based method can still achieve reasonable runtime. During partitioning, we set its allowed number of segments in each partition as 10 for further self-adaptive partitioning methodology.

In the third experiment, we demonstrate the effectiveness of our self-adaptive partitioning methodology for SDP, as shown in Fig. 8. We try different partition granularities (from 5 to 80) for three small test cases, where the maximum number of segments in each partition is limited. From Fig. 8(a) and Fig. 8(b), the average and maximum timing are quite similar, which means that partitioning has 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 performance is insignificant. Furthermore, Fig. 8(c) shows that the run-time increases drastically with the partition granularity. Notably, without self-adaptive partitioning methodology, the number of critical segments to deal with is so high that it takes more than one hour to run even a small benchmark by releasing 0.5%. Therefore, we can see that self-adaptive partitioning methodology benefits the run-time for SDP significantly. Meanwhile, we can observe that when the constraint is set to 10, the run-time can reach its lowest point. Therefore, in our implementation for SDP we set the default partition granularity as 10.

In the last experiment, we further analyze the impact of critical ratio to the performance of SDP based method. Critical ratio is an important parameter to determine how many critical nets are released. In Table 2, we release 0.5% critical nets to see the improvement. Here we evaluate SDP-based method by releasing more critical nets. Meanwhile, we compare the average critical path timing, maximum critical path timing, and run time with TILA for one small benchmark adaptec1. From Fig. 9(a) and Fig. 9(b), we see that the average timing decreases slightly with the increase of critical ratio for both SDP and TILA. However for maximum timing comparison, we see that TILA does not control the maximum timing well. The reason may be that TILA applies a Lagrangian-based relaxation optimization for via capacity constraints, which may affect the timing improvements. In Fig. 9(c), we observe that for SDP-based method the runtime increases in proportion to the critical ratio. This illustrates that our method has a well-controlled scalability.

5. CONCLUSION

This paper targets at optimizing critical path timing during layer assignment stage. First we propose the ILP formulation for the problem, and then present the self-adaptive quadruple partition algorithm to benefit the run-time for SDP. Based on this speed-up algorithm, SDP based method is developed. The experimental results show that our work can outperform TILA

Table 2: Performance Comparison on ISPD'08 Benchmarks

bench	TILA-0.5% [4]					SDP-0.5%				
	Avg(T_{cp}) (10^3)	Max(T_{cp}) (10^3)	OV#	via# (10^5)	CPU(s) (s)	Avg(T_{cp}) (10^3)	Max(T_{cp}) (10^3)	OV#	via# (10^5)	CPU(s) (s)
adaptec1	228.54	4378.58	49121	19.31	85.66	204.88	4205.71	50947	19.26	188.32
adaptec2	101.13	1432.26	43352	19.27	72.99	93.88	1421.68	38478	19.32	158.60
adaptec3	219.88	4613.90	89429	36.74	216.53	209.41	4583.29	92299	36.76	739.73
adaptec4	134.40	5610.61	71275	32.22	180.04	117.43	5590.84	73186	32.44	635.44
adaptec5	275.11	5492.27	98099	55.26	265.94	216.15	5311.75	84537	55.26	723.02
bigblue1	409.88	2596.13	45715	21.70	105.07	322.41	2065.42	46256	21.56	229.29
bigblue2	98.98	10571.84	100530	43.32	123.38	95.58	10728.23	115240	43.49	322.90
bigblue3	29.27	1007.70	78860	52.67	224.18	21.53	373.80	66795	52.92	843.42
bigblue4	39.91	3798.42	107745	110.03	395.86	33.56	3750.95	97148	110.37	1191.26
newblue1	43.13	354.48	56863	22.34	61.26	39.52	343.09	57744	22.44	113.87
newblue2	109.96	6166.35	57889	29.05	102.87	107.85	6130.09	35566	29.25	175.13
newblue4	112.93	5776.33	104657	47.66	182.92	105.53	5395.42	85163	47.73	408.32
newblue5	182.41	2839.57	188697	86.80	419.45	151.41	2771.55	157943	87.00	1952.86
newblue6	159.83	2480.43	113461	78.55	312.79	124.75	2298.74	97859	78.53	736.43
newblue7	31.04	1283.51	175834	163.95	597.62	25.33	1254.22	144580	164.28	2137.26
average	145.09	3893.49	92101.8	54.6	223.10	124.61	3748.32	82916.1	54.7	704.39
ratio	1.00	1.00	1.00	1.00	1.00	0.86	0.96	0.90	1.00	3.16

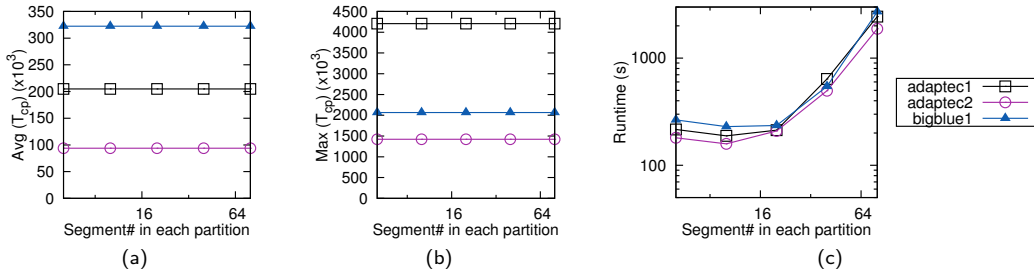


Figure 8: Partition size impact on three small cases. (a) The impact on Avg(T_{cp}); (b) The impact on Max(T_{cp}); (b) The impact on runtime.

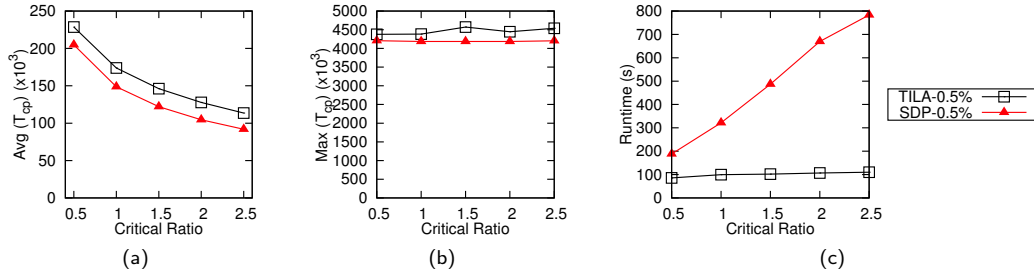


Figure 9: Critical ratio impact on benchmark adaptec1. (a) The impact on Avg(T_{cp}); (b) The impact on Max(T_{cp}); (b) The impact on runtime.

by 14% for average delay and 4% for maximum delay of the critical paths.

Acknowledgment

This work is supported in part by NSF, Oracle, and CUHK Direct Grant for Research.

6. REFERENCES

- [1] J. H.-C. Chen, T. E. Standaert, E. Alptekin, T. A. Spooner, and V. Paruchuri, "Interconnect performance and scaling strategy at 7 nm node," in *Proc. IITC*, 2014, pp. 93–96.
- [2] J. Hu and S. S. Sapatnekar, "A survey on multi-net global routing for integrated circuits," *Integration, the VLSI Journal*, vol. 31, no. 1, pp. 1–49, 2001.
- [3] M.-K. Hsu, N. Katta, H. Y.-H. Lin, K. T.-H. Lin, K. H. Tam, and K. C.-H. Wang, "Design and manufacturing process co-optimization in nano-technology," in *Proc. ICCAD*, 2014, pp. 574–581.
- [4] B. Yu, D. Liu, S. Chowdhury, and D. Z. Pan, "TILA: Timing-driven incremental layer assignment," in *Proc. ICCAD*, 2015, pp. 110–117.
- [5] T.-H. Lee and T.-C. Wang, "Congestion-constrained layer assignment for via minimization in global routing," *IEEE TCAD*, vol. 27, no. 9, pp. 1643–1656, 2008.
- [6] K.-R. Dai, W.-H. Liu, and Y.-L. Li, "Efficient simulated evolution based rerouting and congestion-relaxed layer assignment on 3-D global routing," in *Proc. ASPDAC*, 2009, pp. 570–575.
- [7] W.-H. Liu and Y.-L. Li, "Negotiation-based layer assignment for via count and via overflow minimization," in *Proc. ASPDAC*, 2011, pp. 539–544.
- [8] T.-H. Lee and T.-C. Wang, "Simultaneous antenna avoidance and via optimization in layer assignment of multi-layer global routing," in *Proc. ICCAD*, 2010, pp. 312–318.
- [9] J. Ao, S. Dong, S. Chen, and S. Goto, "Delay-driven layer assignment in global routing under multi-tier interconnect structure," in *Proc. ISPD*, 2013, pp. 101–107.
- [10] M. Queyranne, "Performance ratio of polynomial heuristics for triangle inequality quadratic assignment problems," *Operations Research Letters*, vol. 4, no. 5, pp. 231–234, 1986.
- [11] C.-H. Hsu, H.-Y. Chen, and Y.-W. Chang, "Multi-layer global routing considering via and wire capacities," in *Proc. ICCAD*, 2008, pp. 350–355.
- [12] A. D. Gunawardena, S. Jain, and L. Snyder, "Modified iterative methods for consistent linear systems," *Linear Algebra and its Applications*, vol. 154, pp. 123–143, 1991.
- [13] L. Vandenberghe and S. Boyd, "Semidefinite programming," *SIAM Review (SIREV)*, vol. 38, no. 1, pp. 49–95, 1996.
- [14] Gurobi Optimization Inc., "Gurobi optimizer reference manual," <http://www.gurobi.com>, 2014.
- [15] B. Borchers, "CSDP, a C library for semidefinite programming," *Optimization Methods and Software*, vol. 11, pp. 613–623, 1999.
- [16] "OpenMP," <http://www.openmp.org/>.
- [17] G.-J. Nam, C. Sze, and M. Yildiz, "The ISPD global routing benchmark suite," in *Proc. ISPD*, 2008, pp. 156–159.