Incremental Layer Assignment for Critical Path Timing

Derong Liu, Bei Yu[†], Salim Chowdhury[‡] and David Z. Pan
ECE Dept., University of Texas at Austin; [†]CSE Dept., Chinese University of Hong Kong; [‡]Oracle Corp.

Motivation and Problem

- Layer assignment assigns segments to metal layers after 2-D global routing.

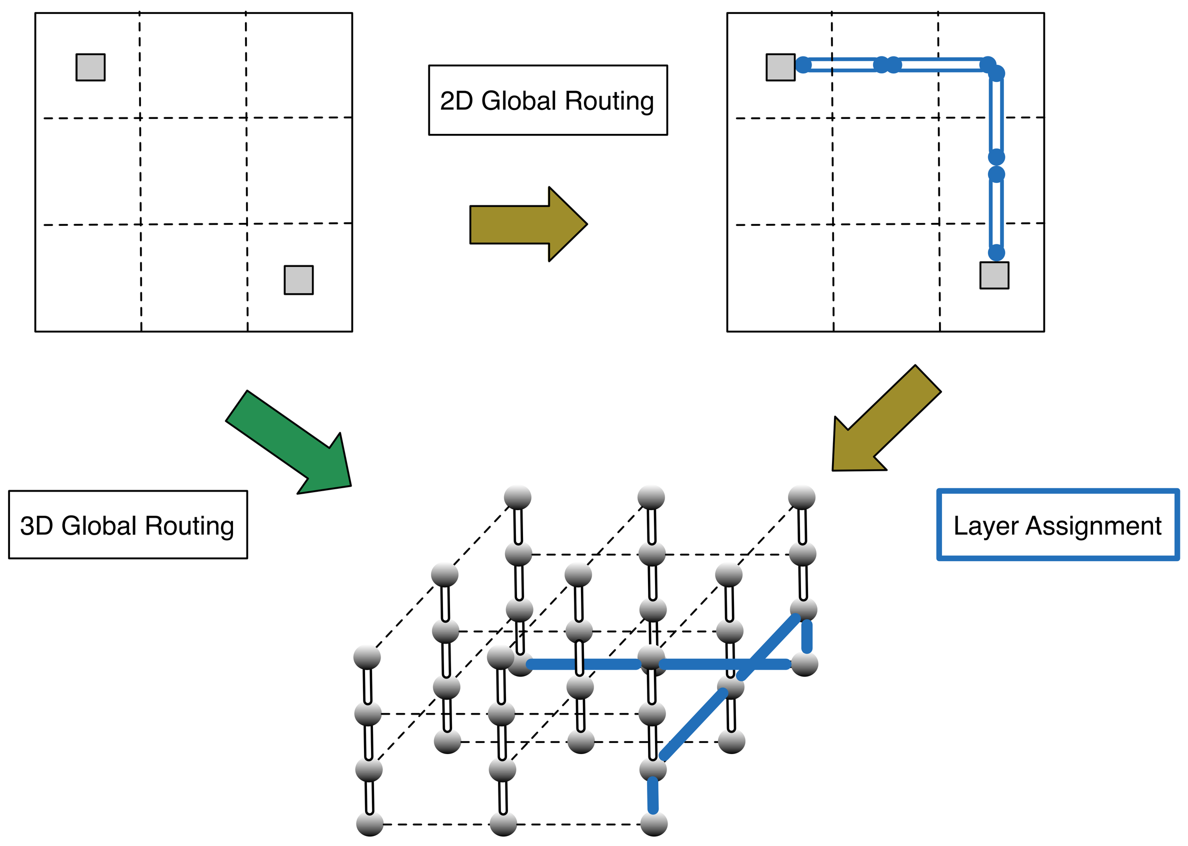
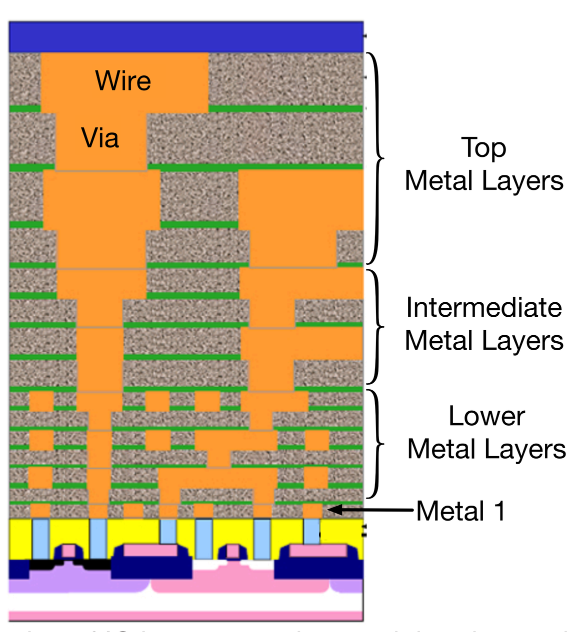


Figure: General Global Routing Flow.

Wires and vias on top layers are wider and much less resistive.



Layer	Resistance
M10	1.00
M9	1.00
M8	3.26
M7	3.26
M6	3.26
M5	3.26
M4	5.58
M3	23.26
M2	19.30
M1	23.26

Figure: The cross section of IC interconnection stack in advanced technology nodes. The normalized resistance values of different metal layers are listed in the table (source: [1] Hsu. et al. ICCAD'2014 [2] Yu. et al. ICCAD'2015).

Timing critical segments prefer higher metal layers.

Non-Critical Nets: n1 n2; Critical Net: n3

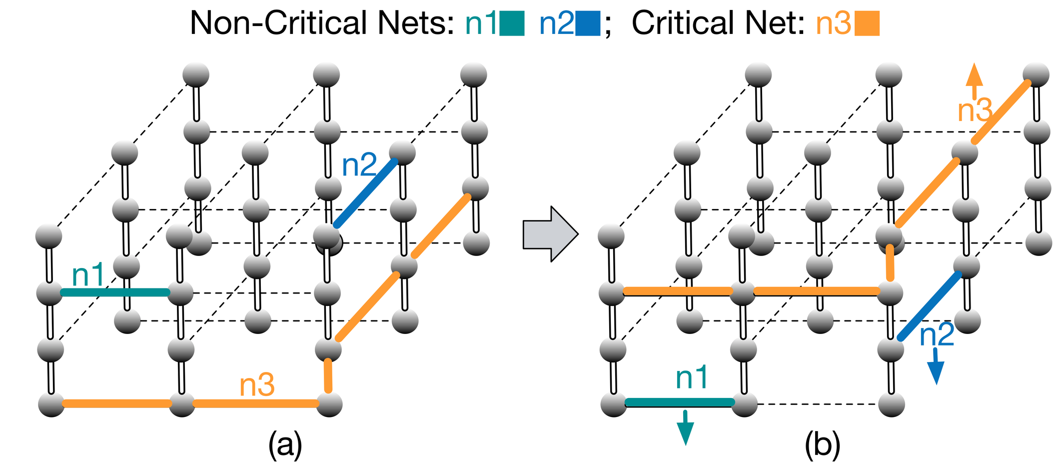
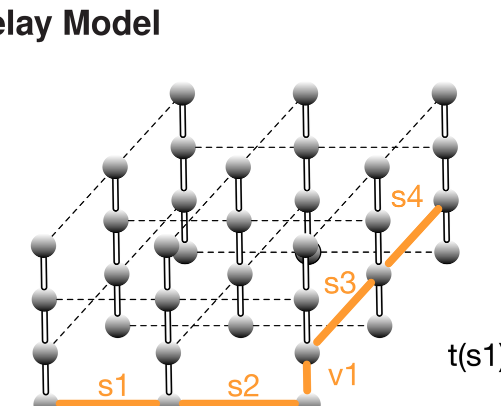


Figure: Incremental layer assignment example, (a) initial assignment, (b) assignment after timing optimization (source: [2] Yu. et al. ICCAD'2015).

Problem: Critical Path Layer Assignment
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.

Model Description

Elmore Delay Model



Maximum Path Delay:
t(s1) + t(s2) + t(s3) + t(s4) + t(v1)

Figure: Net critical path timing considering segment and via delays.

- Segment delay depends on its assigned layer and its downstream capacitance, while via delay depends on the connecting two segments and their assigned layers.

Capacity Constraints

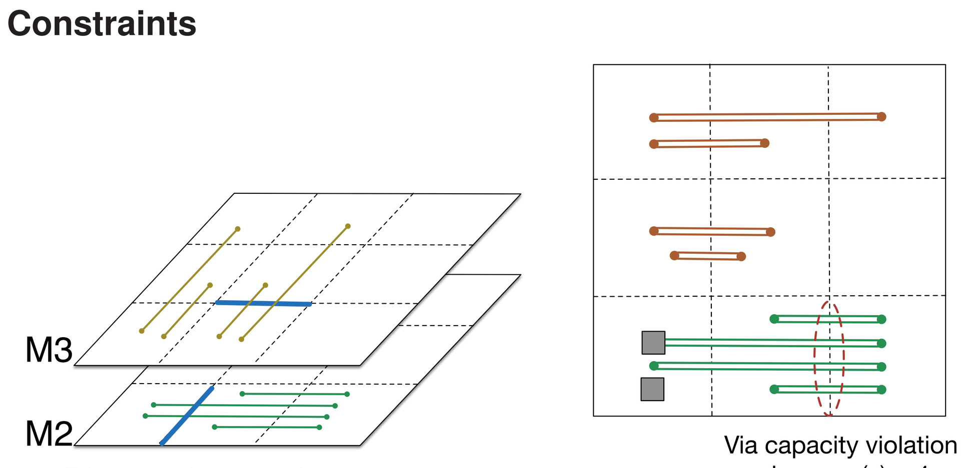


Figure: Example illustrations of capacity constraints, (a) edge capacity, (b) via capacity.

- Edge capacity constraint – the maximum number of routing tracks along the edge.
- Via capacity constraint – the maximum number of allowable vias in this grid.

CPLA Algorithms

ILP Formulation

$$\min \sum_{i\in S(Nc)}\sum_{j}^{L} t_s(i,j)\cdot x_{ij} + \sum_{i,p\in Sx(Nc)}\sum_{j}^{L}\sum_{q}^{L} t_v(i,j,p,q)\cdot y_{ijpq}$$

Binary variables: x_{ij} represents segment i assigned on layer j;
y_{ijpq} represents the via connecting segment i and p from layer j to q, which is equal to the product of x_{ij} and x_{pq}.

Constraints
Each released segment has to be assigned.

$$\sum_{j} x_{ij} = 1; \qquad \forall i \in S(Nc) \quad j \in L$$

Edge capacity constraint:

$$\sum_{i\in S(e)} x_{ij} \le cap_e(j); \qquad \forall e \in E$$

Via capacity constraint:

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

Self-adaptive Quadruple Partition

***K* × *K* Partition supports parallel scheme.**

- Incremental layer assignment can be solved in each partition separately.
- Multithreading is applied to provide further speed-up.
- The most recent updated results by peer threads can be taken into accounts.

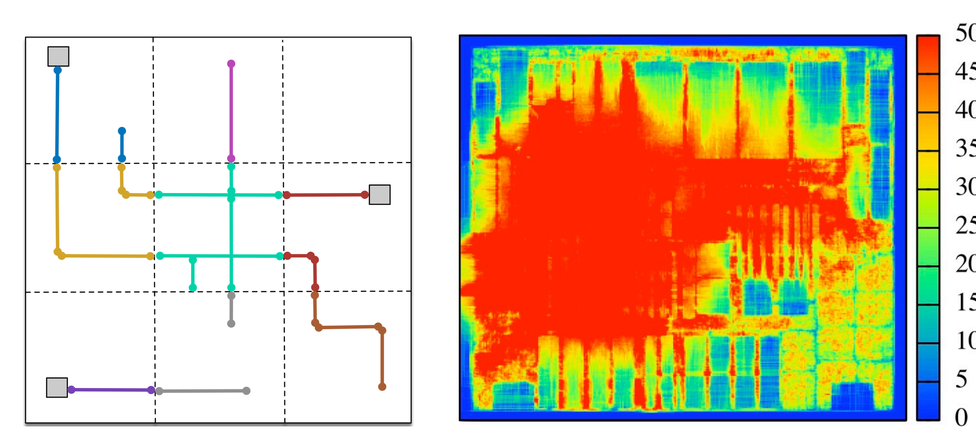


Figure: Example of grid partition, (a) nets partition, (b) routing density for benchmark adaptec1 by NCTU-GR.

Self-adaptive Quadruple Partition

- Uniform division by $K \times K$ may lead to unbalanced resource allocation.
- Self-adaptive partition provides similar number of segments in each partition.
- Through multi-threading, each thread deals with a workload in a well-balanced manner.

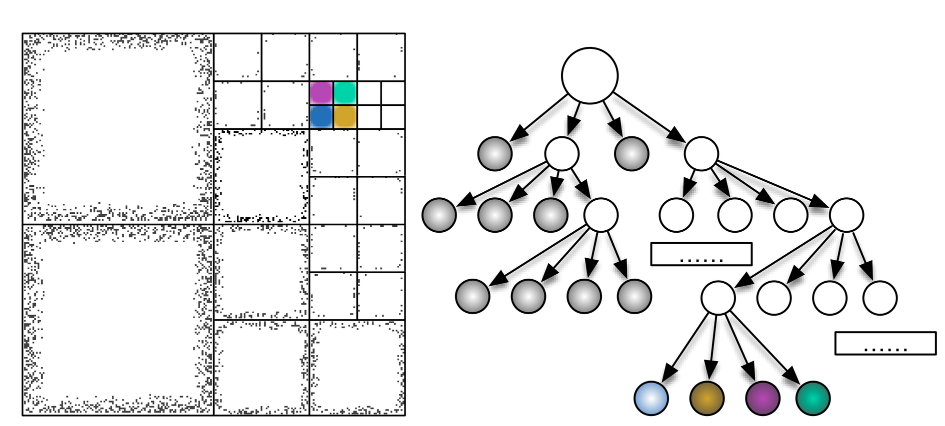


Figure: Sub-grid partition illustration, (a) sub-grid partition, (b) sub-grid partition tree.

Semidefinite Programming Relaxation

The proposed self-adaptive partition provides an opportunity for further speed-up. Semidefinite programming (SDP) is solvable in polynomial time while providing a theoretically better solution than Linear Programming (LP).

- The objective function:

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

- Matrix T - $|S\cdot L|$-dimensional symmetric matrix representing timing costs
- Matrix X - $|S\cdot L|$-dimensional symmetric matrix representing variables

$$T = \begin{pmatrix} t_s(i,j) & \dots t_v(i,j,p,q) \\ \dots & \dots \quad \dots \\ t_v(i,j,p,q)\dots & t_s(p,q) \end{pmatrix}, \qquad X = \begin{pmatrix} x_{ij} & \dots y_{ijpq} \\ \dots & \dots \dots \\ y_{ijpq}\dots & x_{pq} \end{pmatrix}.$$

Semidefinite Programming Example

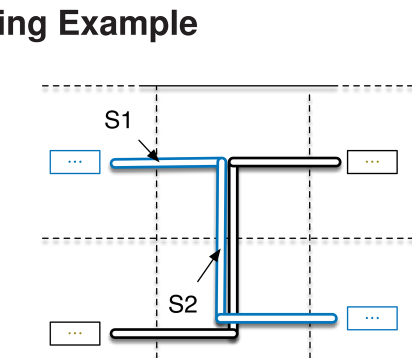


Figure: Example of layer assignment through solving SDP.

- Cost matrix T and solution matrix X of this example

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

- Segment $S2$ overlaps with other segments, resulting in continuous solutions.
- Therefore, post mapping is required to provide integer solutions.

Overall Algorithm Flow

Post mapping transfers continuous solutions into discrete assignment result.

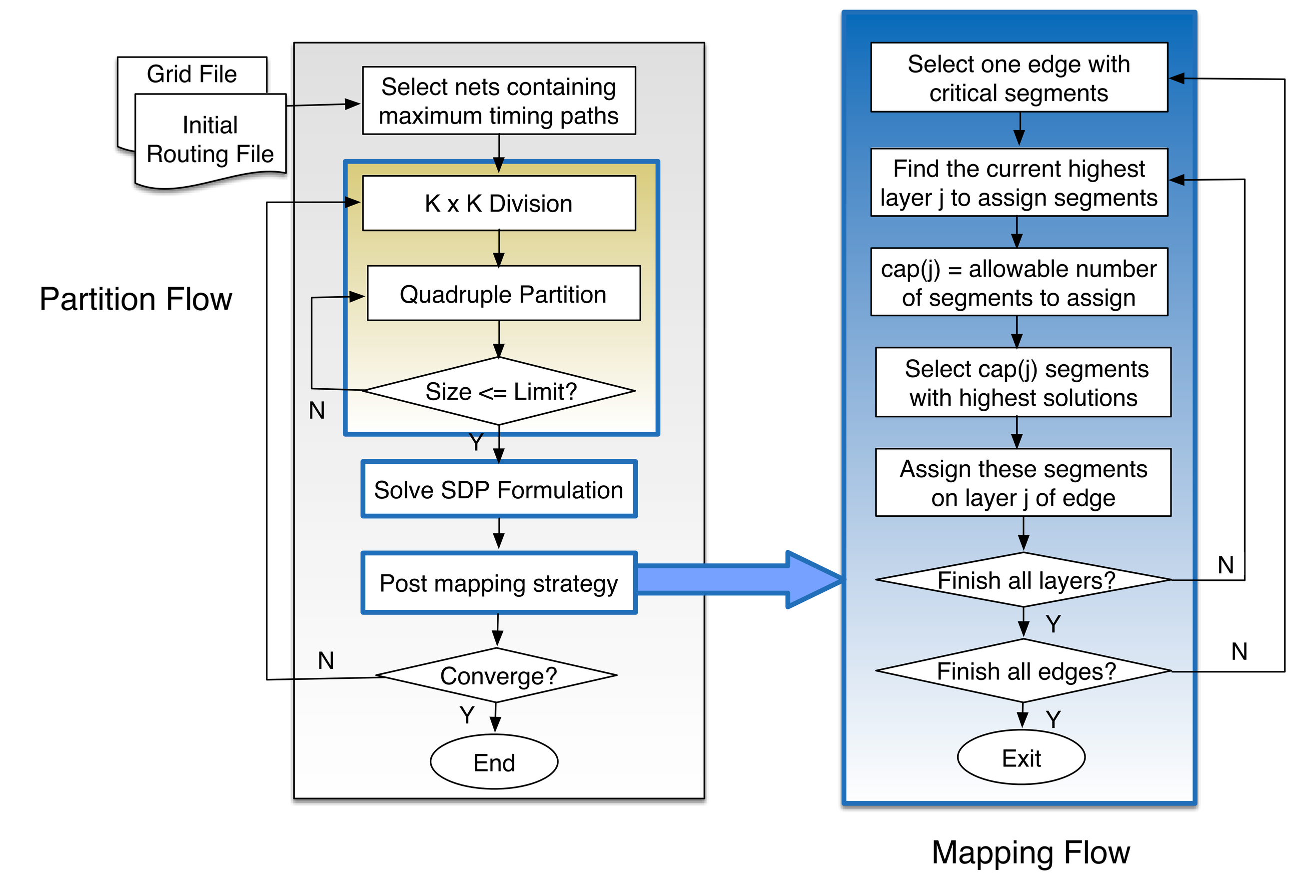


Figure: Iterative incremental layer assignment algorithm flow.

Experimental Results

Experimental Setup

- CPLA implemented in C++, Gurobi as the MILP solver and CSDP as the SDP solver
- Linux machine with 2.9GHz Inter(R) Core and 192GB memory
- Initial routing and layer assignment result from NCTU-GR and NVM tool

Comparison on layer assignment result with TILA-0.5%

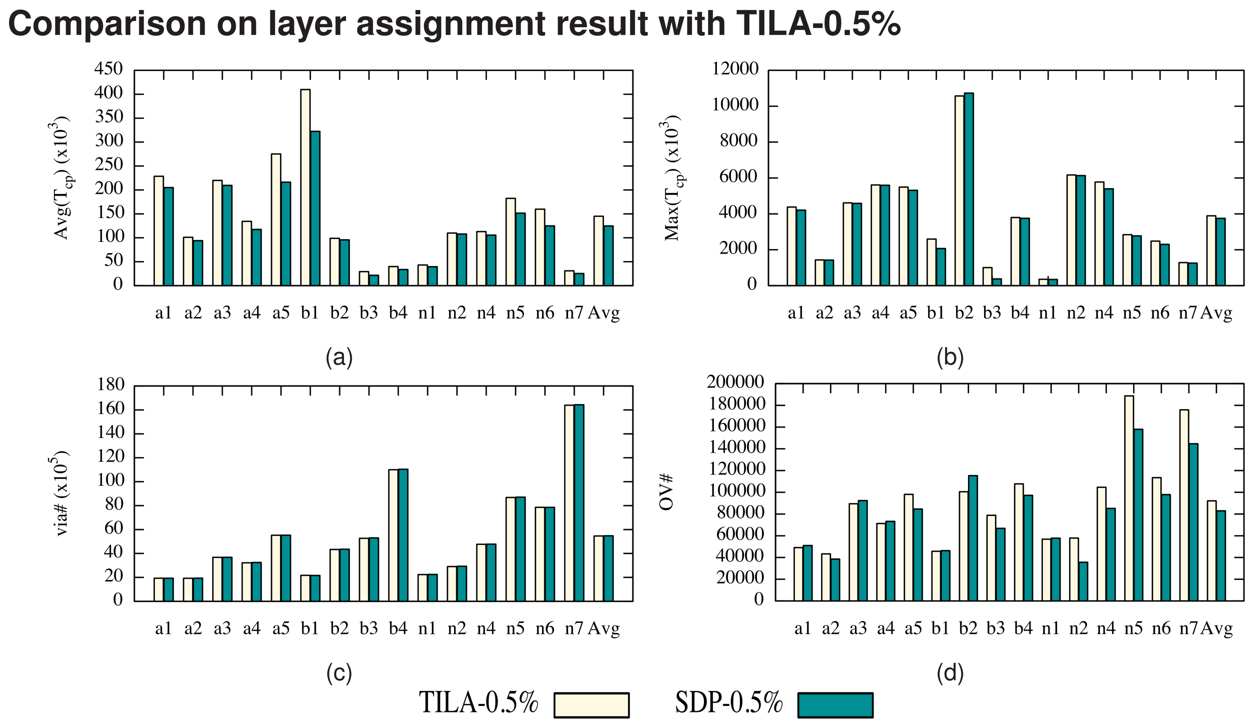


Figure: Comparison among different strategies, (a) $Avg(T_{cp})$, (b) $Max(T_{cp})$, (c) via number, (d) via violations.

- Average delay improved by 14% while maximum delay improved by 4%.
- Via violations improved by 10% with the similar number of vias.

Performance comparison with ILP

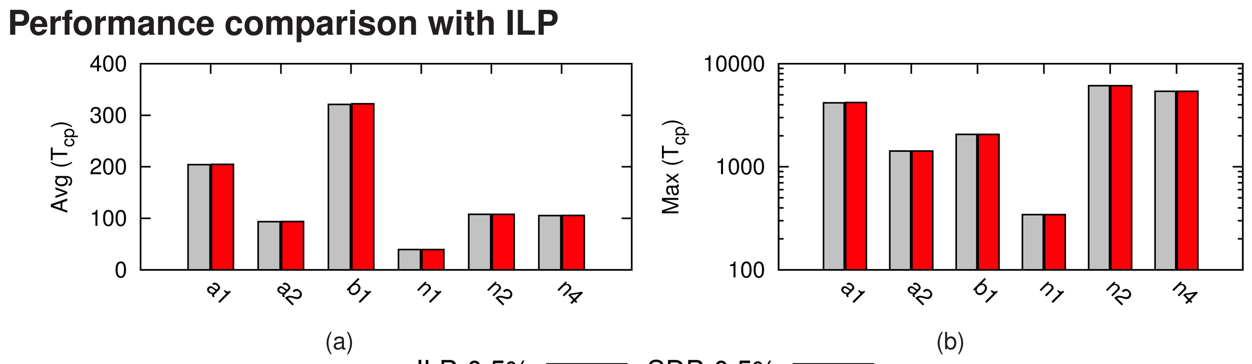


Figure: Performance comparison with ILP, (a) $Avg(T_{cp})$, (b) $Max(T_{cp})$.

- Both average delay and maximum delay of SDP achieve the similar performance.

Run-time comparison

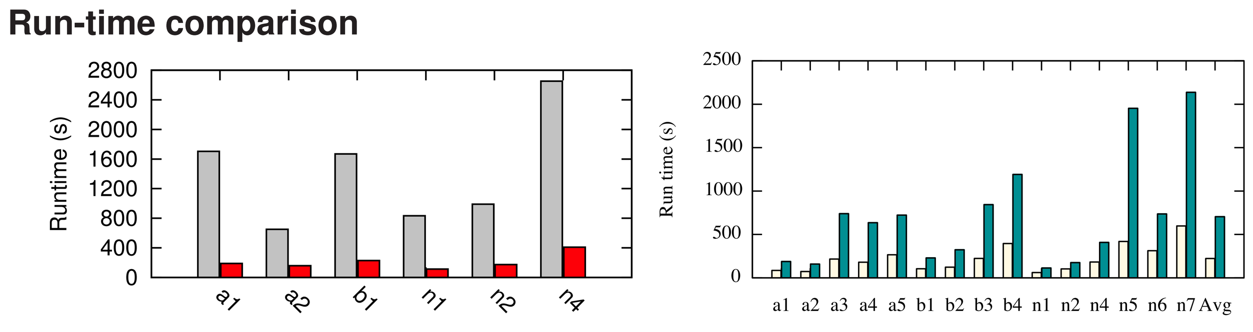


Figure: Run time comparison among different strategies, (a) with ILP, (b) with TILA.

- The run-time of SDP is lower than ILP but slower than TILA [2] (Yu. et al. ICCAD'2015).

Partition size impact on performance

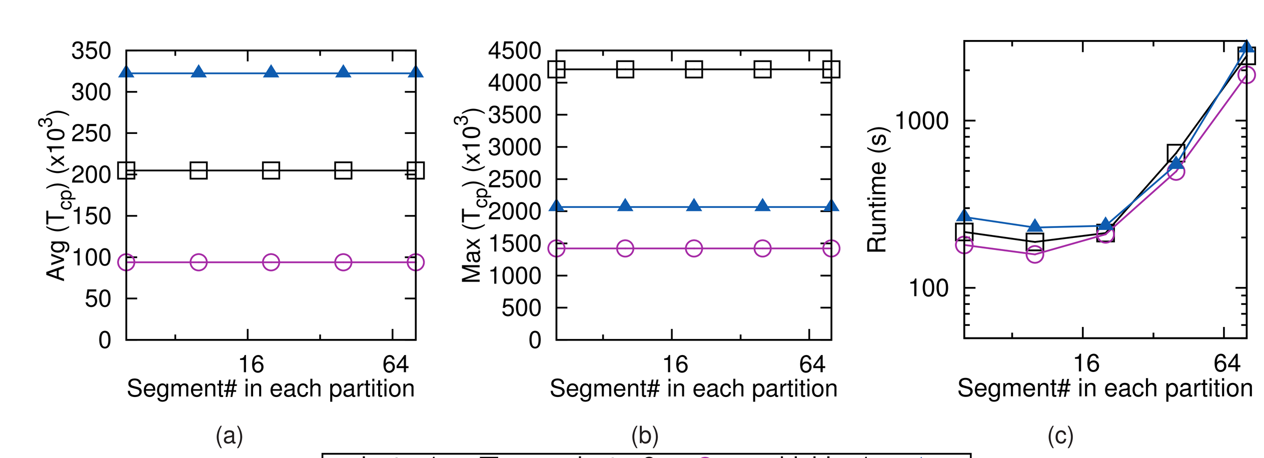


Figure: Comparison among different partition sizes, (a) $Avg(T_{cp})$, (b) $Max(T_{cp})$, (c) run time.

- For different partition sizes, similar performance can be achieved.
- When partition size is equal to 10, the lowest run-time can be obtained.

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

Derong Liu – ECE Department – University of Texas at Austin – Austin, USA
E-Mail: derongliu@utexas.edu
WWW: http://www.cerc.utexas.edu/utda