

Physical Design Summer School

2019 Beijing

Placement

Evangeline F.Y. Young

The Chinese University of Hong Kong

Outline

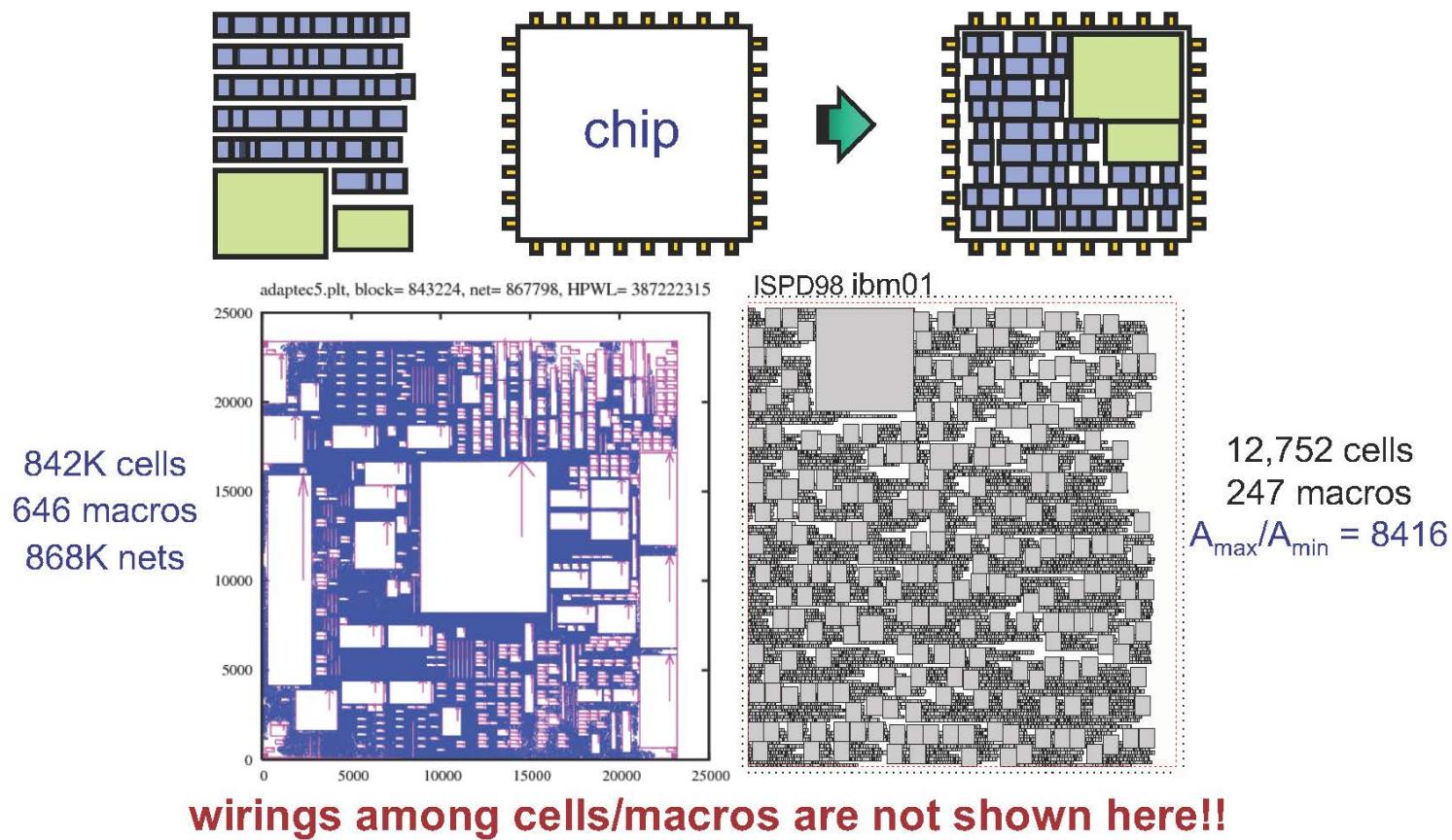
- ASIC Placement and Basics
- FPGA Placement
- AI Engine Accelerated Placer
- Detailed Placement with Multi-row Cells

Outline

- ASIC Placement and Basics
- FPGA Placement
- AI Engine Accelerated Placer
- Detailed Placement with Multi-row Cells

VLSI Placement

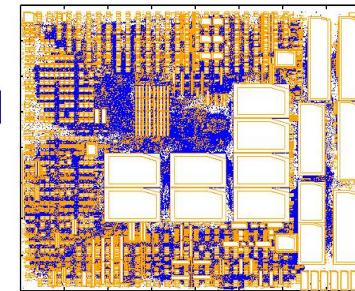
- Place objects into a die s.t. no objects overlap with each other & some cost metric (e.g., wirelength) is optimized



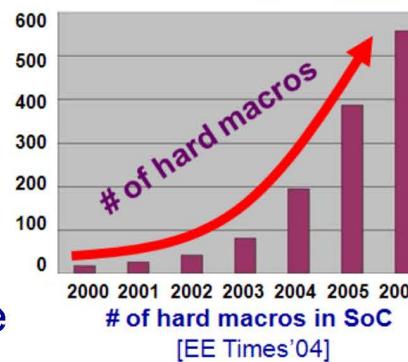
Source from Prof. Yaowen Chang of NTU @ Summer School 2018

Modern Placement Challenges

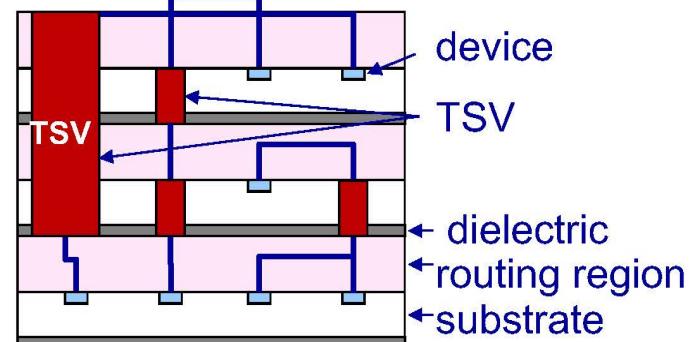
- High complexity
 - Millions of objects to be placed
- Placement constraints
 - Preplaced blocks
 - Routability/density
 - Regions
- Mixed-size placement
 - Hundreds/thousands of large macros with millions of small standard cells
- Many more (technology, etc.)
 - 3D IC, DFM, FinFET, etc.



2.5M
placeable
objects
mixed-size
design

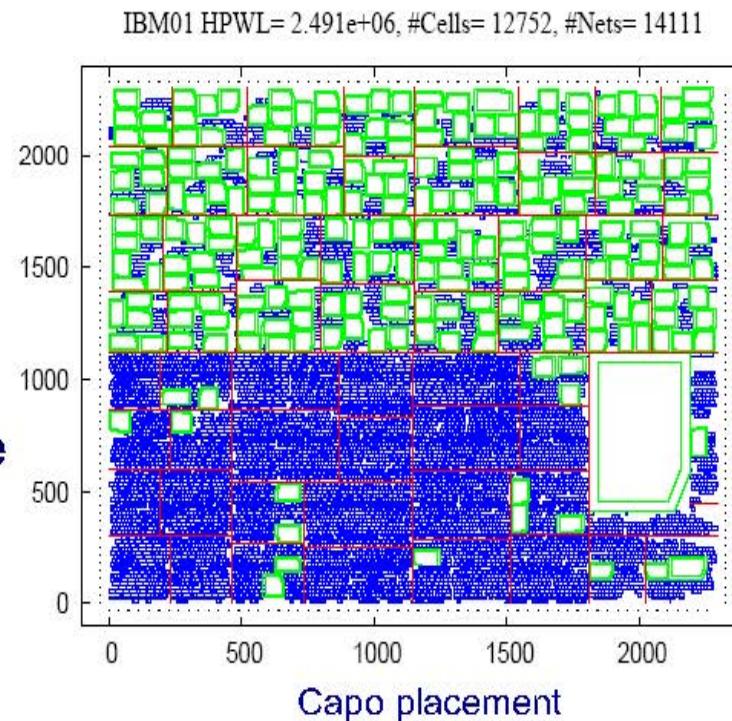


Macros have
revolutionized
SoC design

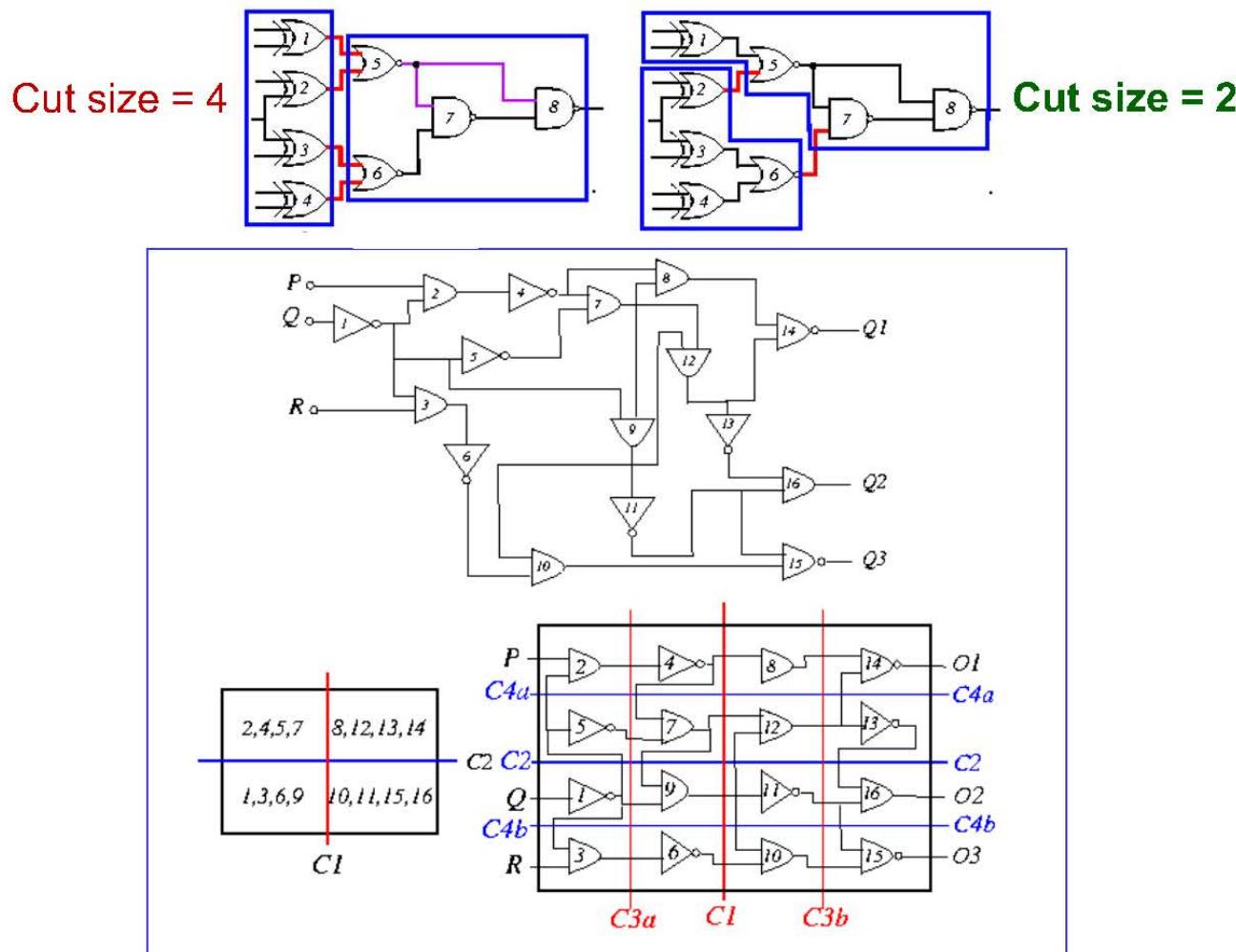


Placement Method 1: Partitioning

- Partition a circuit into subregions while minimizing crossing interconnections
- Pros
 - Fast
 - Good scalability for large-scale designs
- Cons
 - Harder to handle large white spaces
 - Lower quality



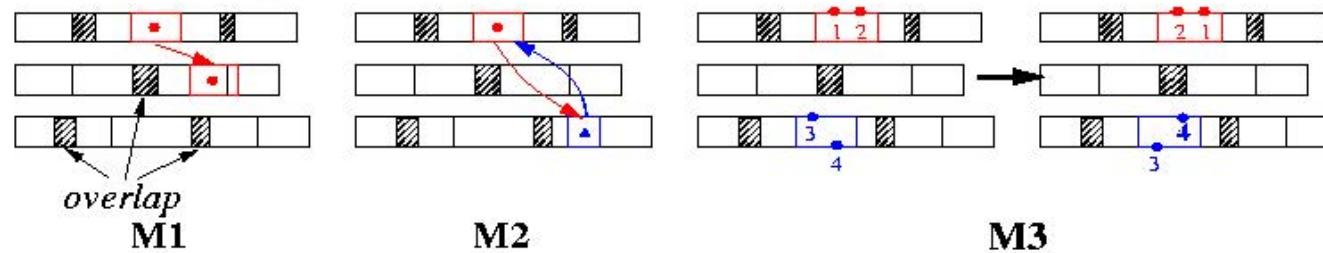
Partitioning Placement Example



Source from Prof. Yaowen Chang of NTU @ Summer School 2018

Method 2: Simulated Annealing (SA)

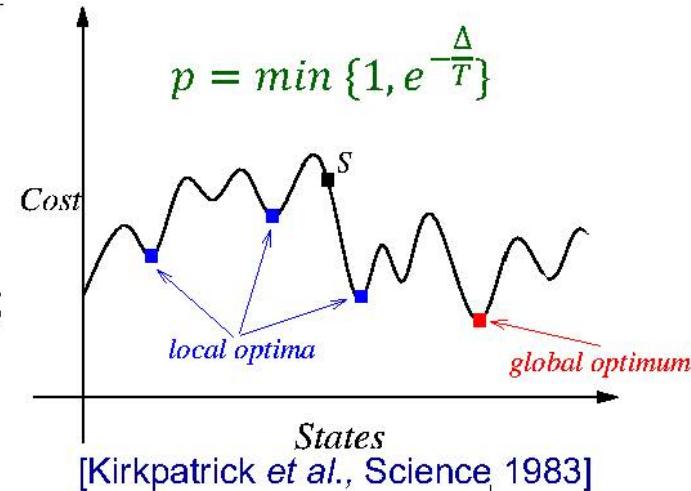
- Pros: flexible; Cons: Slow, hard to scale
- Example moves to find desired solutions
 - M_1 : Displace a module to a new location.
 - M_2 : Interchange two modules.
 - M_3 : Change the orientation of a module.



- Cost function: $C = C_1 + C_2 + C_3$.
 - C_1 : total estimated wirelength.
 - C_2 : penalty function for module overlaps.
 - C_3 : penalty function that controls the row length.

Generic Simulated Annealing Algorithm

```
1 begin
2 Get an initial solution  $S$ ;
3 Get an initial temperature  $T > 0$ ;
4 while not yet “frozen” do
5   for  $1 \leq i \leq P$  do
6     Pick a random neighbor  $S'$  of  $S$ ;
7      $\Delta \leftarrow \text{cost}(S') - \text{cost}(S)$ ;
     /* downhill move */
8     if  $\Delta \leq 0$  then  $S \leftarrow S'$ 
     /* uphill move */
9     if  $\Delta > 0$  then  $S \leftarrow S'$  with probability  $e^{-\frac{\Delta}{T}}$  ;
10     $T \leftarrow rT$ ; /* reduce temperature */
11  return  $S$ 
12 end
```



[Kirkpatrick et al., Science 1983]

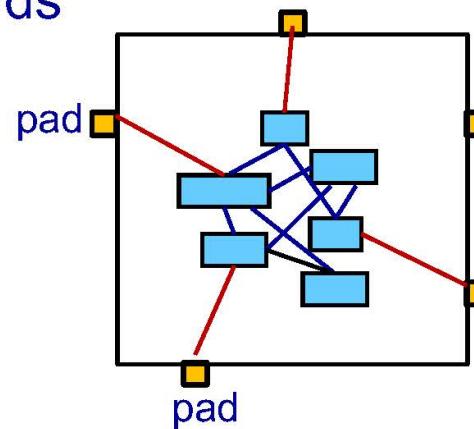
- Use non-zero probability for “up-hill” move to escape from a local optimum & search for a global optimum
 - Probability (p) depends on the cost change (Δ) and total search time (temperature T)

Method 3: Analytical Placement

- Key: Solve a relaxed mathematical programming problem “optimally,” ignoring overlaps (to be fixed later)

$\min \quad W(x, y)$ // objective function (e.g., wirelength)
subject to a set of placement constraints

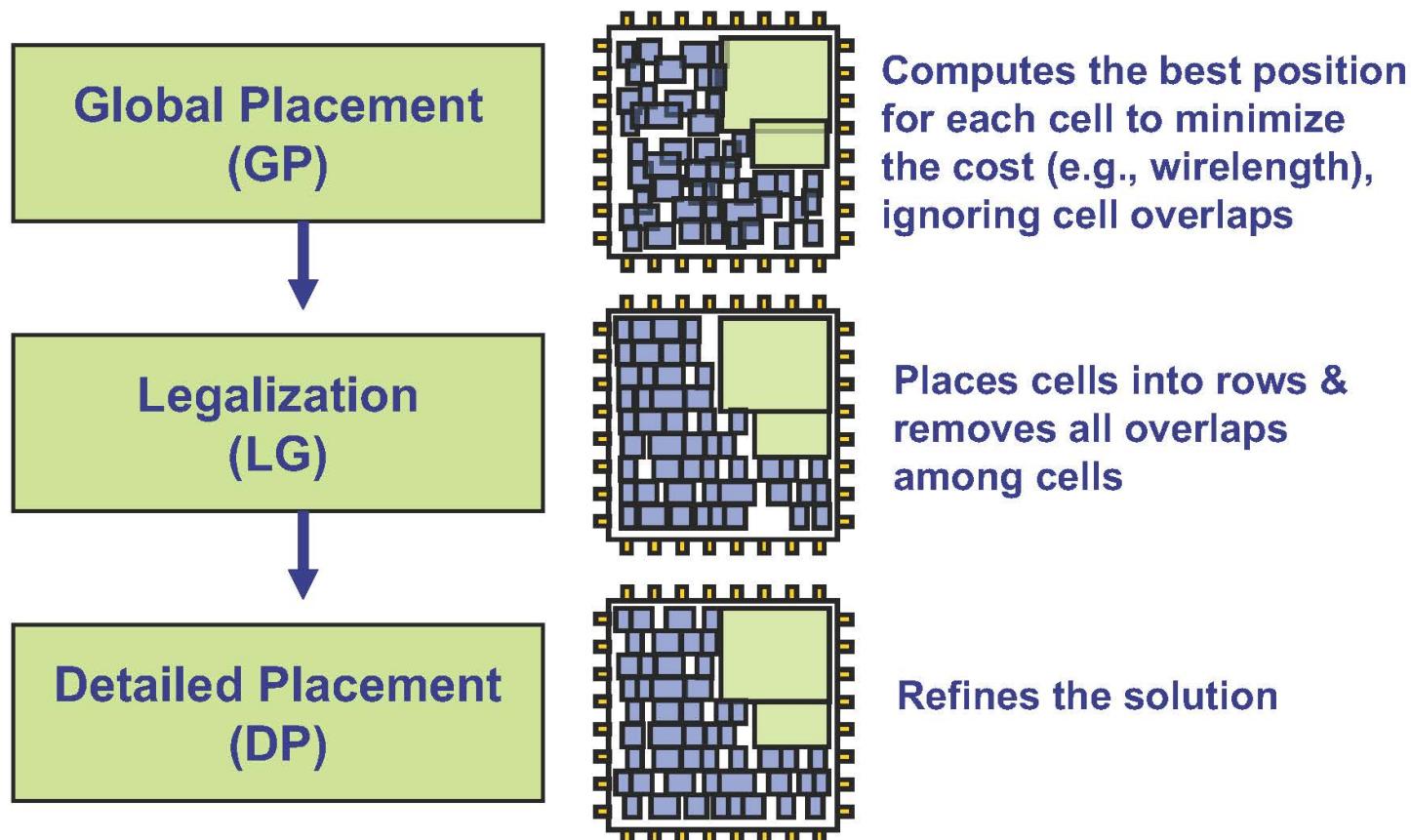
- Adopt a differentiable objective function
- Need pads to pull cells outwards
- Pro: Excellent quality
- Cons: hard to handle big macro
rotation & legalization, etc.



io pad一定要先擺放好，才能透過analytical的方式進行擺放

NTUPlace3/4 Placement Flow

- Chen, et al., “A high quality **analytical placer** considering preplaced blocks and density constraint,” ICCAD-06 (TCAD-08)

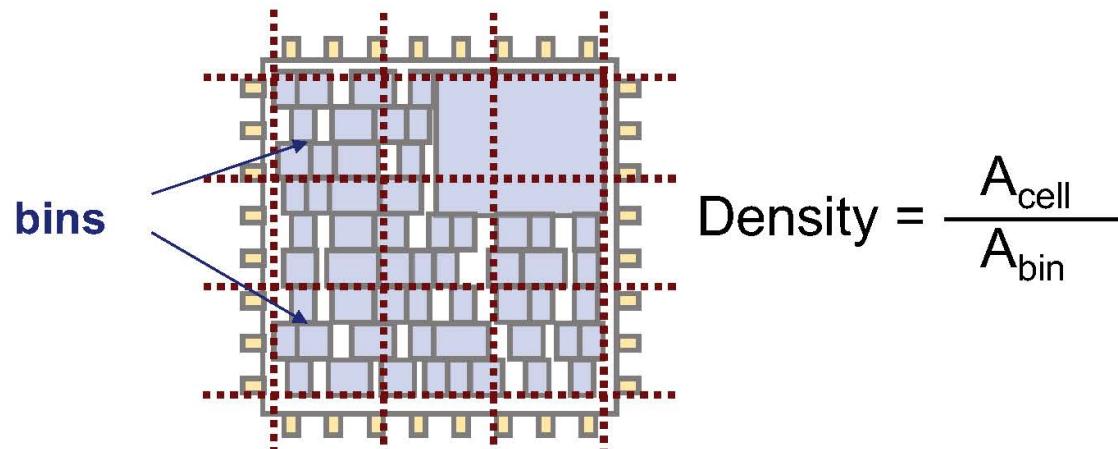


Placement with Density Constraint

- Given a chip region and cell dimensions, divide the placement region into bins
- Determine (x, y) for all movable cells

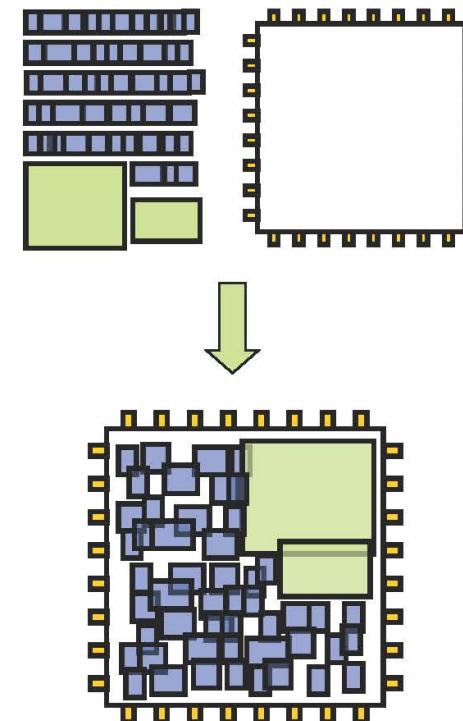
min $W(x, y)$ // wirelength function

s.t. 1. $\text{Density}_b(x, y) \leq \text{MaximumDensity}_b$
for each bin b
2. No overlap between cells



Global Placement

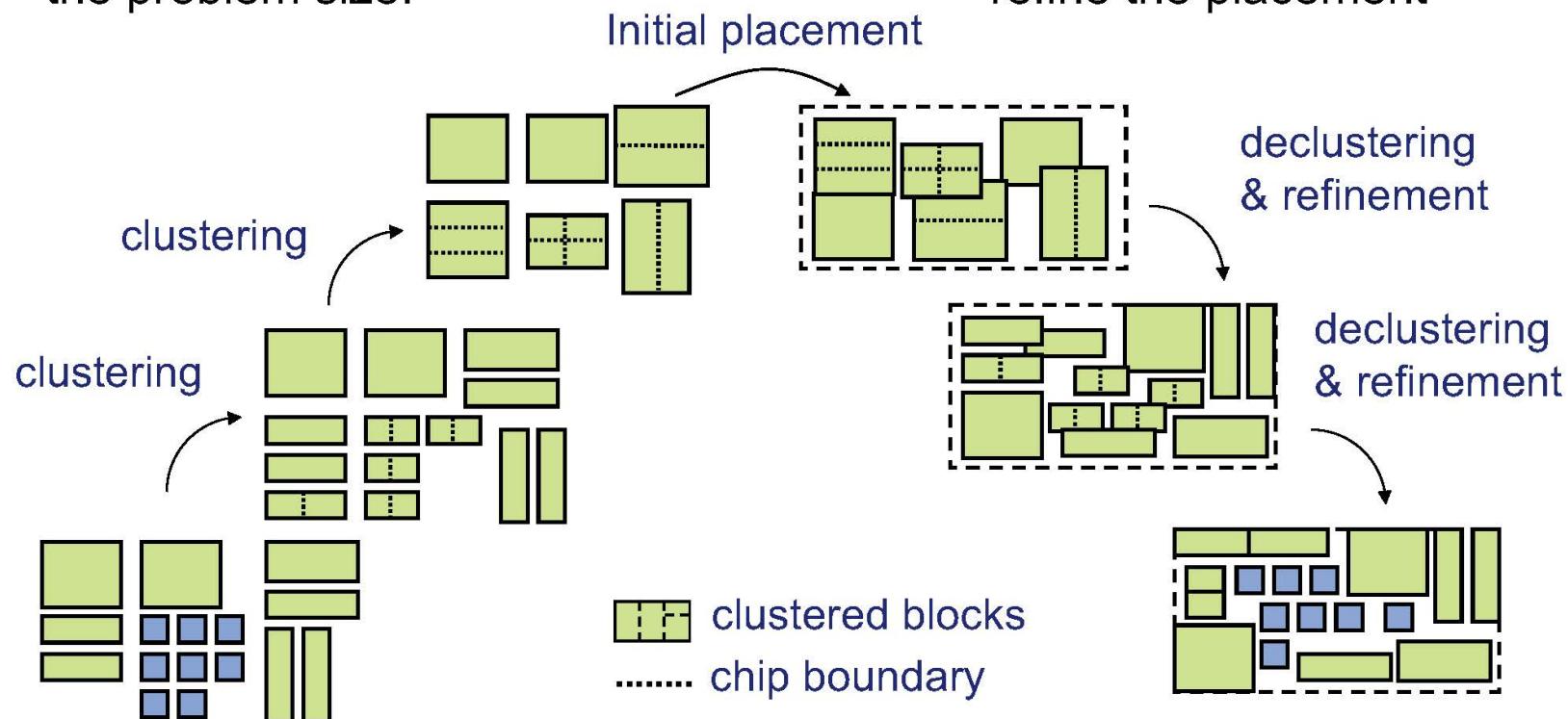
- Placement flow
 - ***Global placement***
 - ***Multilevel framework***
 - ***Analytical formulation with a nonlinear objective function***
 - ***Smoothing techniques for preplaced blocks***
 - ***Free-space allocation for density control***
 - ***Many more....***
 - Legalization
 - Detailed placement



Multilevel Global Placement

Cluster cells based on connectivity/area to reduce the problem size.

Iteratively decluster the clusters and further refine the placement



Analytical Placement Model

- Analytical placement during declustering
- Global placement problem (allow overlaps)

$$\begin{aligned} \min \quad & W(x, y) \\ \text{s.t.} \quad & D_b(x, y) \leq M_b \end{aligned}$$

Minimize wirelength
 D_b : density for bin b
 M_b : max density for bin b

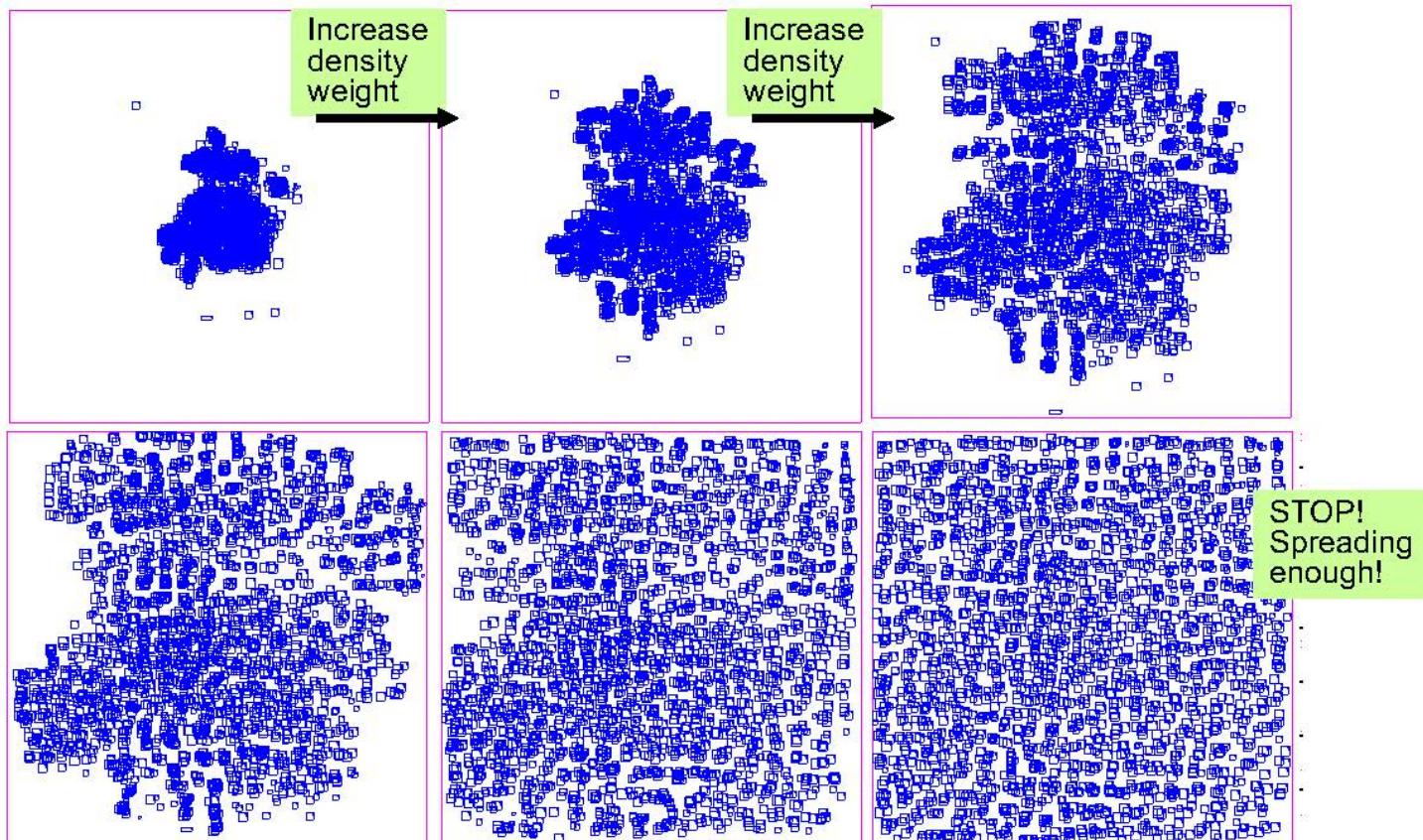
- Relax the constraints into the objective function

$$\min \quad W(x, y) + \lambda \sum (D_b(x, y) - M_b)^2$$

一开始先不考慮constraint帶來的影響。
隨著iteration次數增加再慢慢加大
constraint所造成的penalty

- Use gradient search for it, which needs a smooth & differentiable function
- Increase λ gradually to minimize the density penalty to find cell positions (x, y) with desired wirelength

Optimization Process



Gradient Solver

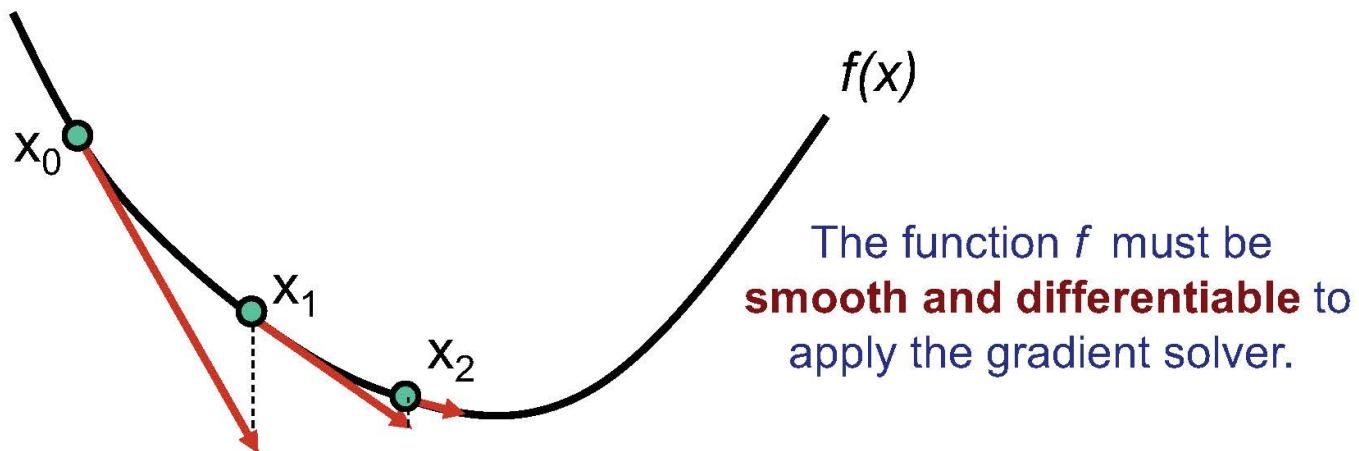
$$\min f(x)$$

[Gradient Solver]

$x_0 \leftarrow$ initial value

Repeat until convergence

$$x_{i+1} = x_i - f'(x)|_{x=x_i} * \text{stepsize}$$

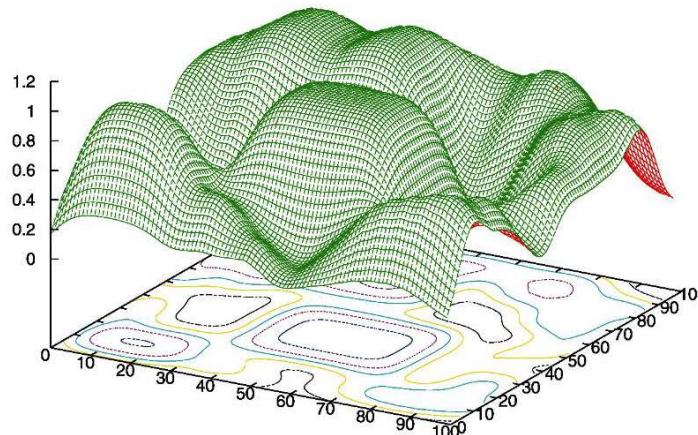


Two Ingredients in Analytical Formulation

$$\min W(x, y) + \lambda \Sigma(D_b(x, y) - M_b)^2$$

↑ ↑
Wirelength Density

Both functions must be smooth & differentiable!!

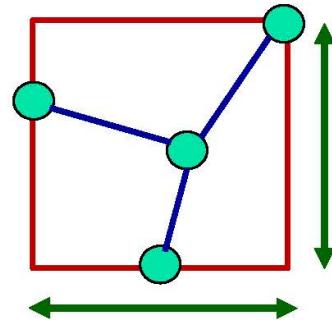


HPWL Wirelength Model

- **Golden:** Half-perimeter wirelength (HPWL) model

$$W(\mathbf{x}, \mathbf{y}) = \sum_{e \in E} (\max_{v_i, v_j \in e} |x_i - x_j| + \max_{v_i, v_j \in e} |y_i - y_j|)$$

- Is not smooth or differentiable!! 因為裡面有絕對值
- Approximations: quadratic, L_p -norm, log-sum-exp, CHKS wirelength models, etc.



Popular Wirelength Models

HPWL
$$W(\mathbf{x}, \mathbf{y}) = \sum_{\text{net } e} (\max_{v_i, v_j \in e} |x_i - x_j| + \max_{v_i, v_j \in e} |y_i - y_j|)$$

quadratic
$$\frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \gamma_{ij} [(x_i - x_j)^2 + (y_i - y_j)^2]$$

Log-sum-exp
$$W_{LSE}(\mathbf{x}, \mathbf{y}) = \gamma \sum_{e \in E} (\ln \sum_{v_k \in e} \exp(x_k / \gamma) + \ln \sum_{v_k \in e} \exp(-x_k / \gamma) + \ln \sum_{v_k \in e} \exp(y_k / \gamma) + \ln \sum_{v_k \in e} \exp(-y_k / \gamma))$$

L_p-norm
$$\sum_{e \in E} ((\sum_{v_k \in e} x_k^p)^{\frac{1}{p}} - (\sum_{v_k \in e} x_k^{-p})^{-\frac{1}{p}} + (\sum_{v_k \in e} y_k^p)^{\frac{1}{p}} - (\sum_{v_k \in e} y_k^{-p})^{-\frac{1}{p}})$$

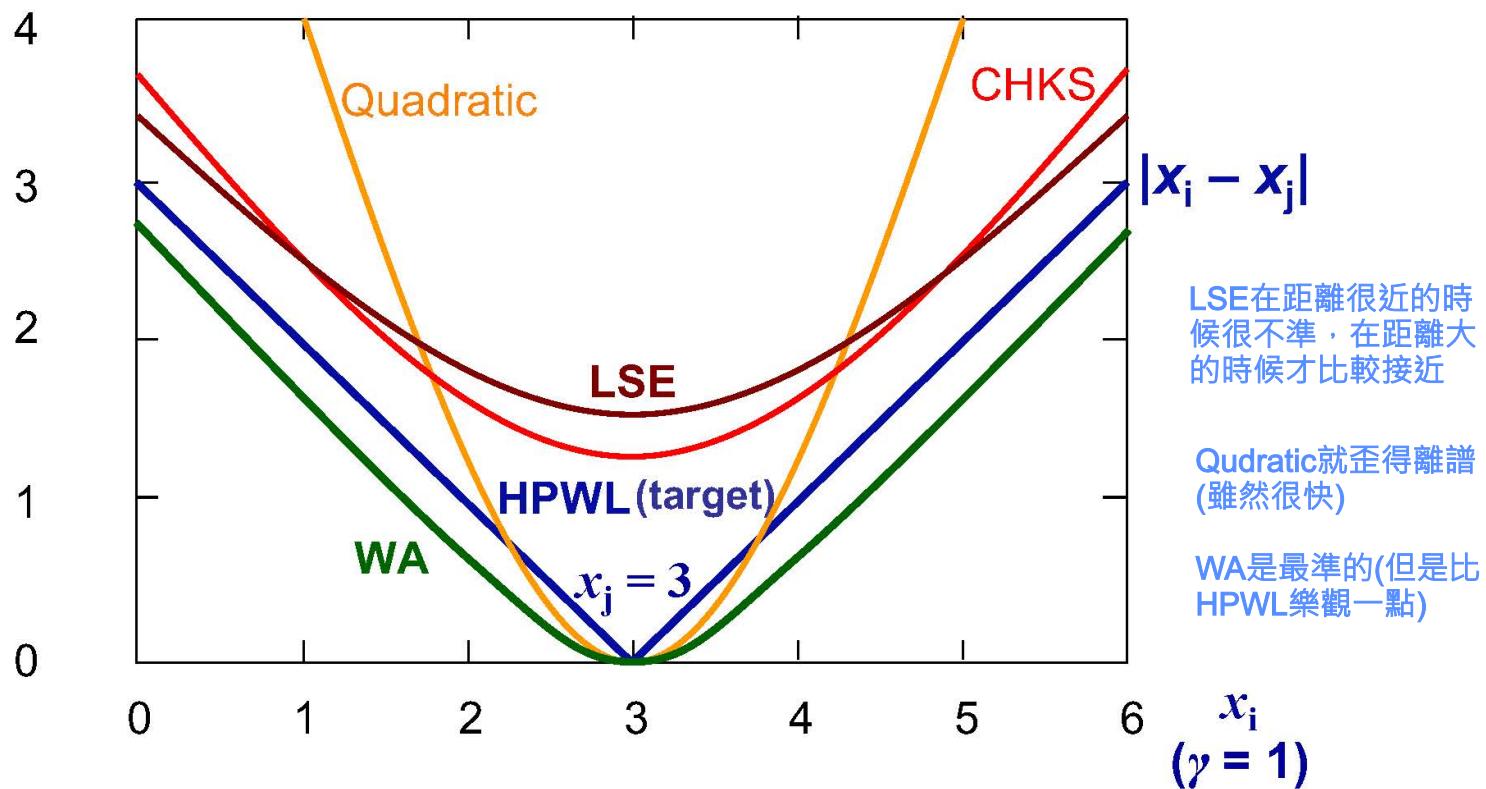
CHKS
$$CHKS(x_1, x_2) = \frac{\sqrt{(x_1 - x_2)^2 + t^2} + x_1 + x_2}{2},$$

Weighted-average
$$\sum_{e \in E} \left(\frac{\sum_{v_i \in e} x_i \exp(x_i / \gamma)}{\sum_{v_i \in e} \exp(x_i / \gamma)} - \frac{\sum_{v_i \in e} x_i \exp(-x_i / \gamma)}{\sum_{v_i \in e} \exp(-x_i / \gamma)} + \frac{\sum_{v_i \in e} y_i \exp(y_i / \gamma)}{\sum_{v_i \in e} \exp(y_i / \gamma)} - \frac{\sum_{v_i \in e} y_i \exp(-y_i / \gamma)}{\sum_{v_i \in e} \exp(-y_i / \gamma)} \right).$$

Popular Wirelength Model Comparisons

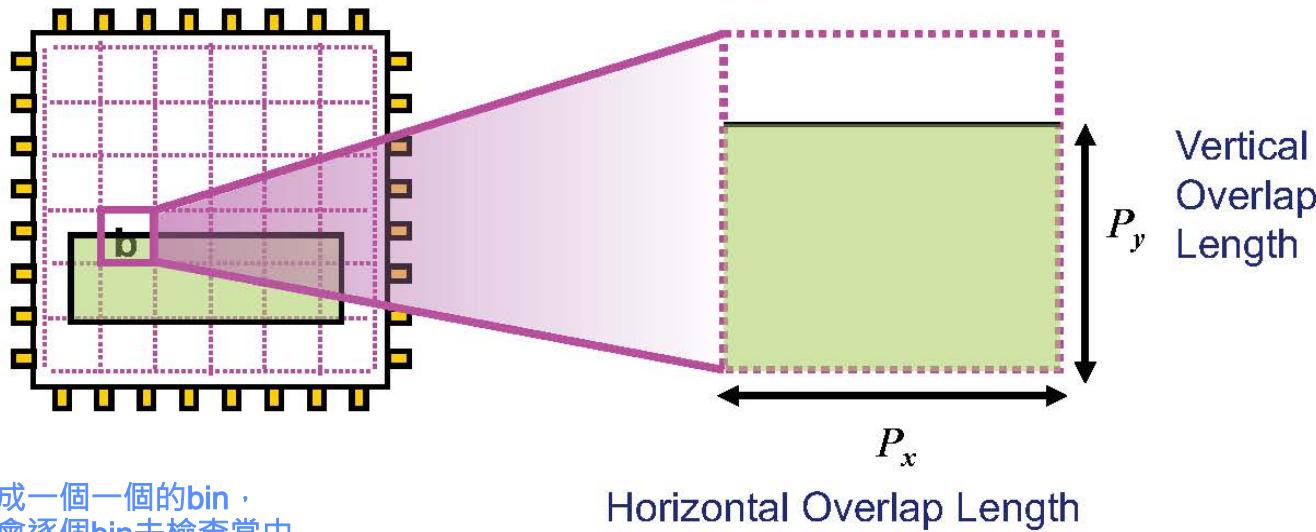
wirelength

functions with 2 variables



Density Model

- Compute the block area in each bin to obtain the bin density

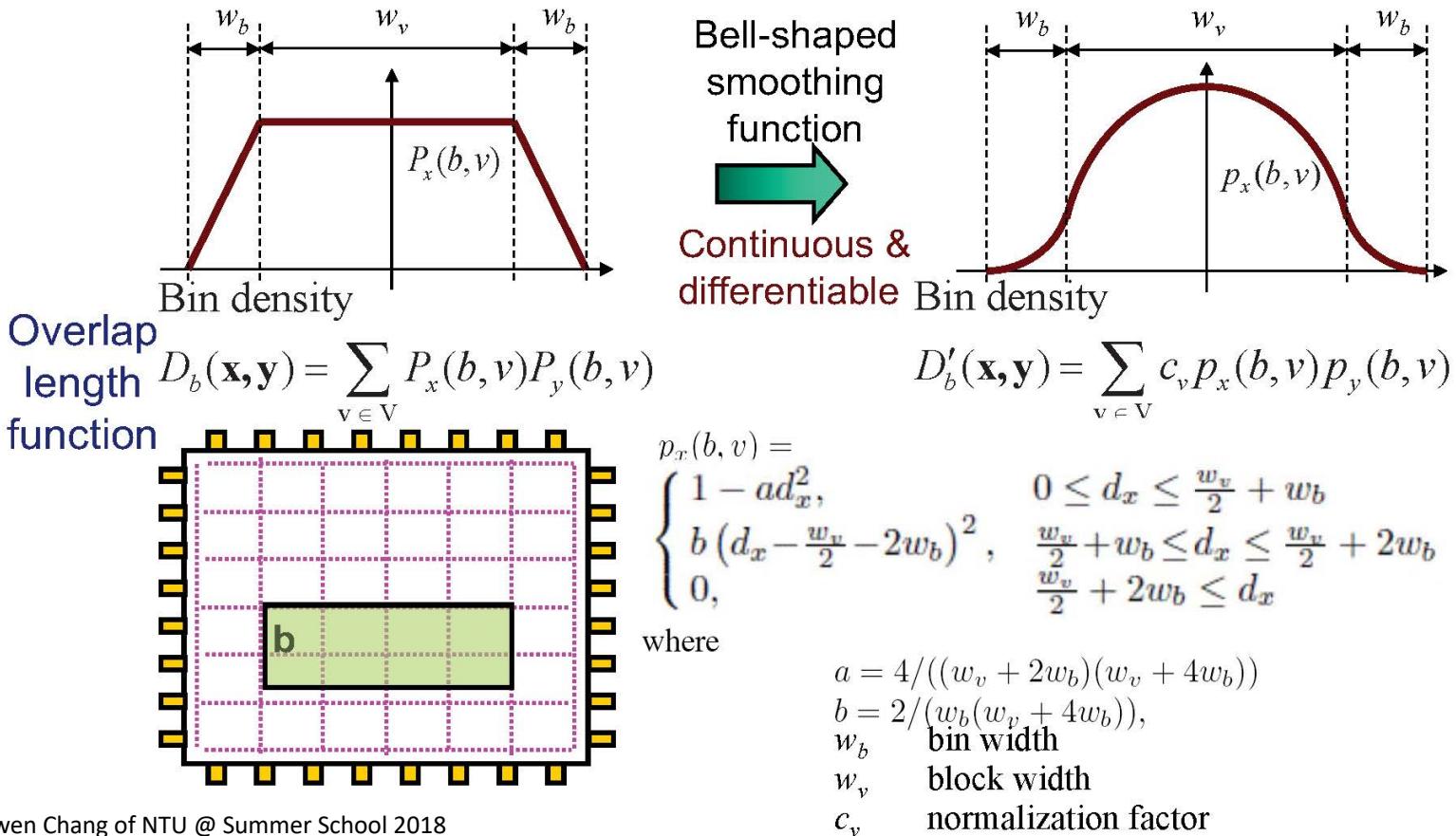


Bin density

$$D_b(\mathbf{x}, \mathbf{y}) = \sum_{v \in V} P_x(b, v)P_y(b, v)$$

Density Smoothing: Bell-Shaped Function

- Could apply the bell-shaped function to make bin density function smooth [Kahng & Wang, ICCAD-04]

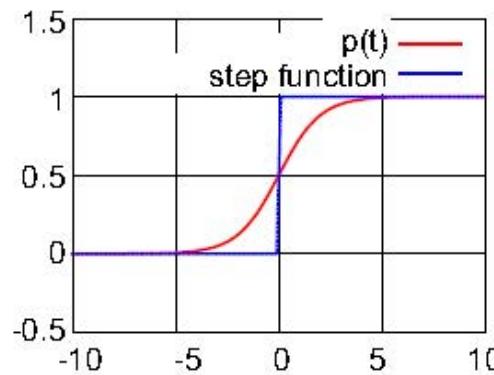


Source from Prof. Yaowen Chang of NTU @ Summer School 2018

Sigmoid Function Smoothing

- Logistic function

$$p(t) = \frac{1}{1 + e^{-\alpha t}}$$



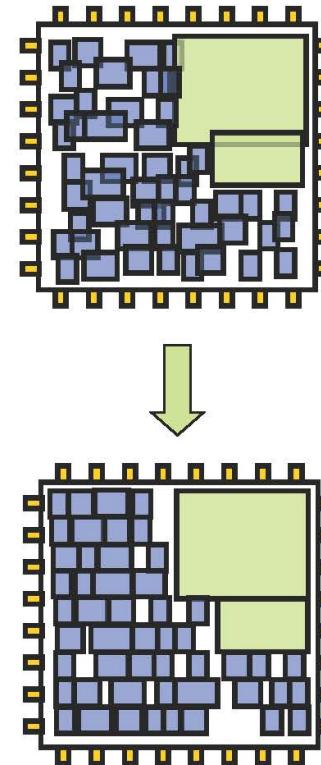
- Smoothed 0-1 logic function

$$f(l, x, u) = \begin{cases} 1, & \text{if } l < x < u \\ 0, & \text{otherwise} \end{cases} \quad \xrightarrow{\hspace{1cm}} \quad p(t) = \frac{1}{1 + e^{-\alpha t}} \quad f(l, x, u) \cong p(x-l)p(u-x)$$

- Is an effective **smooth and differentiable** approximation for the 0-1 logic function
- Approximates exact 0-1 logic function when $\alpha \rightarrow 0$

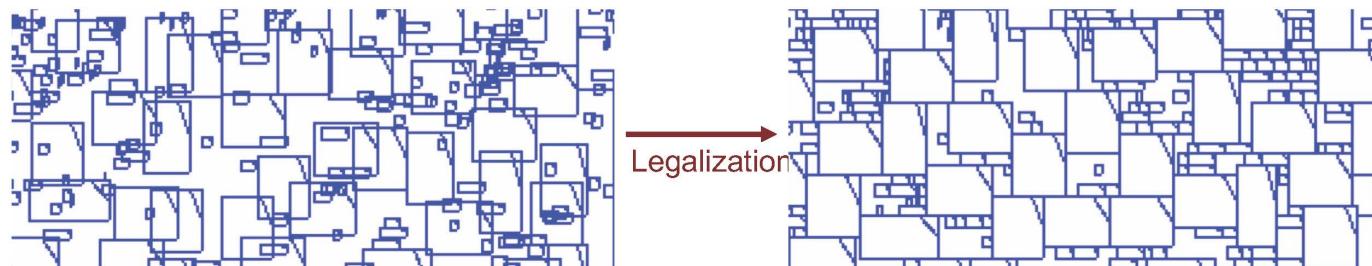
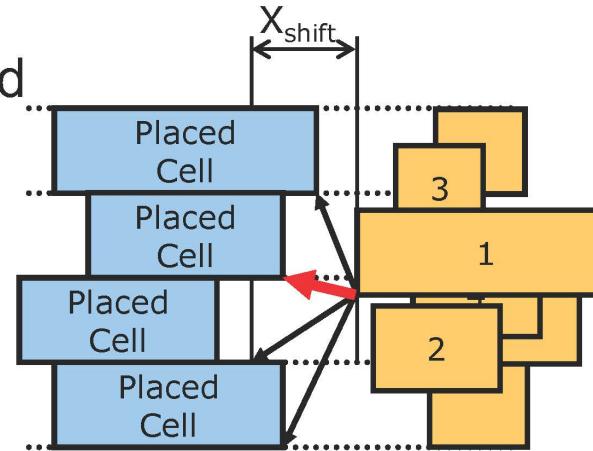
Legalization

- Placement flow
 - Global placement
 - **Legalization**
 - **Mixed-size legalization**
 - **Tetris vs. Abacus**
 - **Look-ahead legalization**
 - Detailed placement



Mixed-Size Legalization

- Determine cell legalization sequence by the x coordinate and cell size
 - Priority = $k_1 x_i + k_2 w_i + k_3 h_i$
 - x_i : x coordinate of block i
 - $w_i(h_i)$: the width (height) of cell i
 - Larger cells are legalized earlier
- Place cell at the position with the smallest displacement within a given range



Abacus Legalizer

- Given a cell ordering, legalization can be formulated as quadratic programming (QP)
- QP is time-consuming for large-scale problems
 - Interior-point method / active-set method: $O((n + m)^3) / O((n + m)^2)$ per iteration (n : #variables; m : #constraints)
- Instead, Abacus uses dynamic programming for cell legalization
 - Spindler et al., ISPD'08

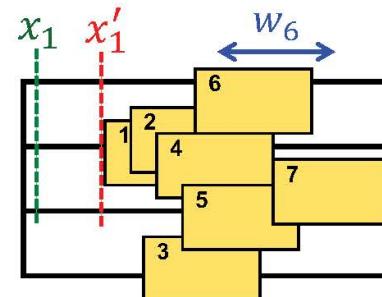
$$\text{QP: } \min \sum_{i=1}^N e_i (x_i - x'_i)^2 \\ \text{s.t. } x_i \geq x_{i-1} + w_{i-1}$$

e_i : weight

x_i : final cell position

x'_i : initial cell position

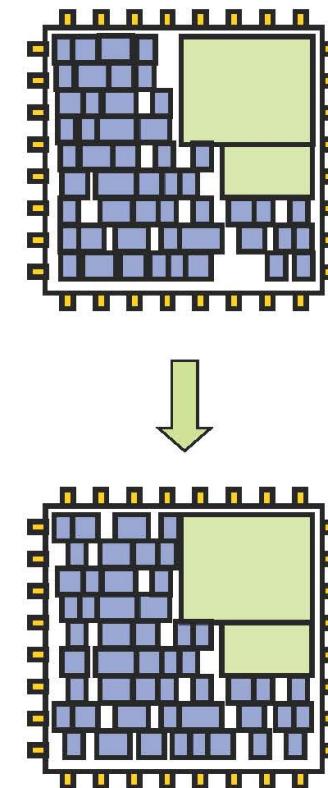
w_i : cell width



**Abacus: dynamic programming
smaller displacement than Tetris**

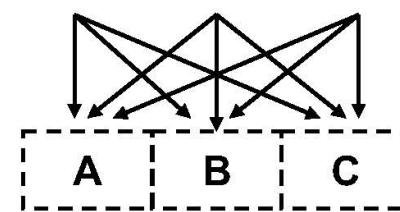
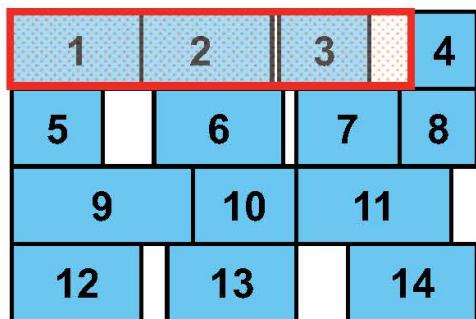
Detailed Placement

- Placement flow
 - Global placement
 - Legalization
 - *Detailed placement*
 - *Cell matching for wirelength minimization*
 - *Cell sliding for density optimization*



Cell Matching

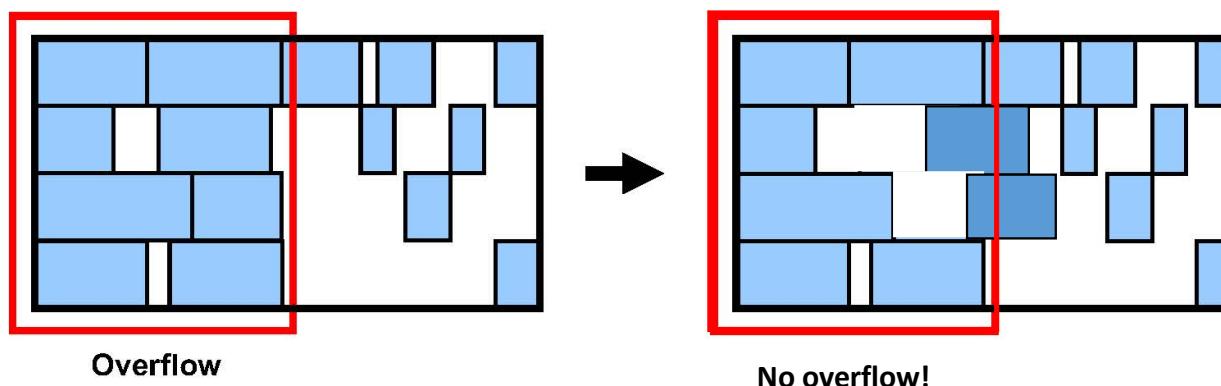
- For wirelength minimization
- Steps
 1. Select a window
 2. Select cells from the window
 3. Create a bipartite matching problem (edge weight = wirelength)
 4. Find the minimum weighted matching to optimize the wirelength
 5. Update cell positions
- Handle **200-300** cells at one time
 - Compared to branch-and-bound which can handle only 6 cells at one time due to its high time complexity



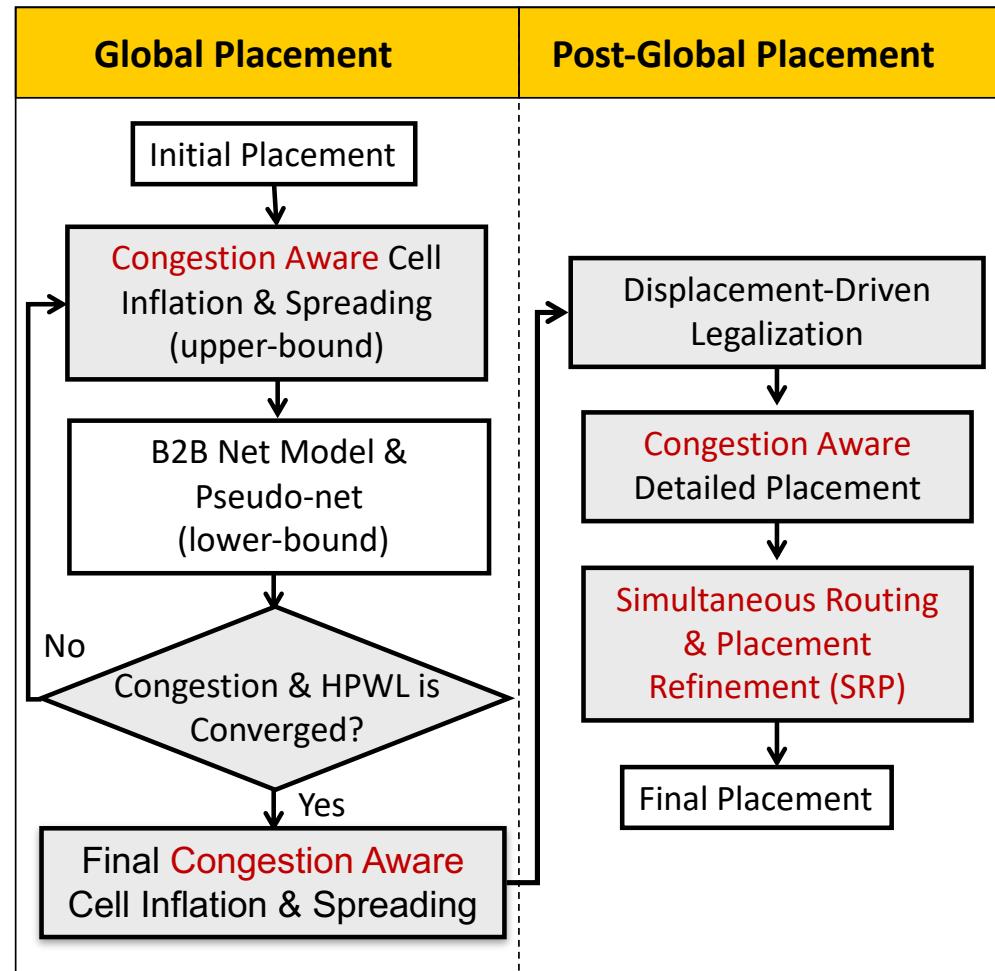
Assign cells {1,2,3} to locations {A,B,C}

Cell Sliding

- Slide the cells from denser regions to sparser ones while preserving the cell order to optimize density
- Steps
 1. Select an overflow window
 2. Calculate the amount of area to shift
 3. Move cells left/right to reduce the density
- Is fast and effective

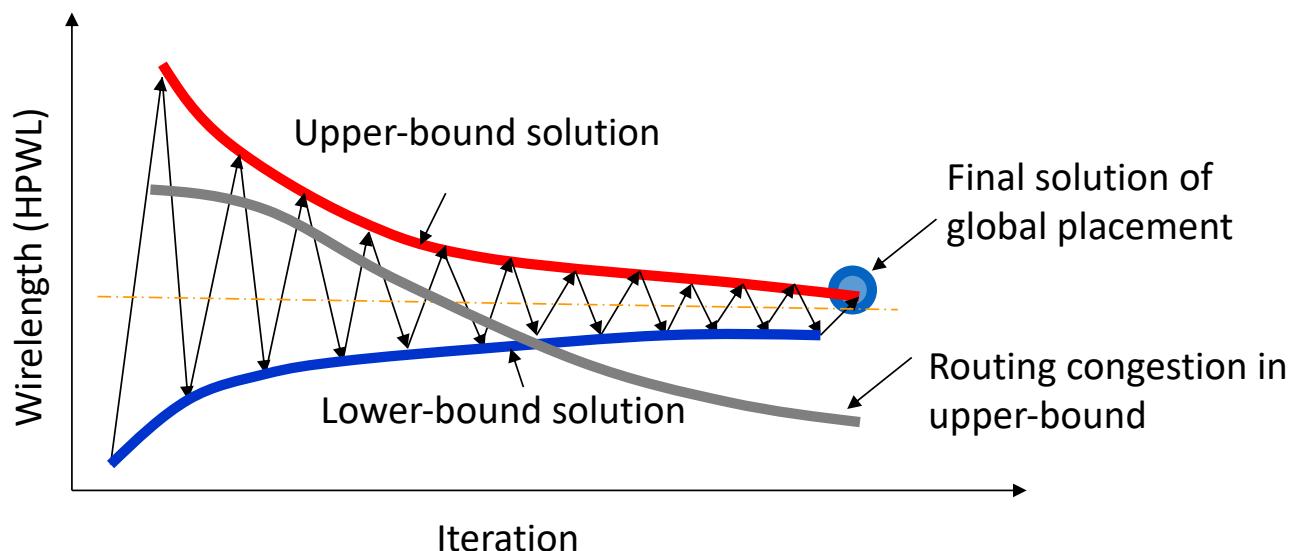


Ripple – Routability Driven



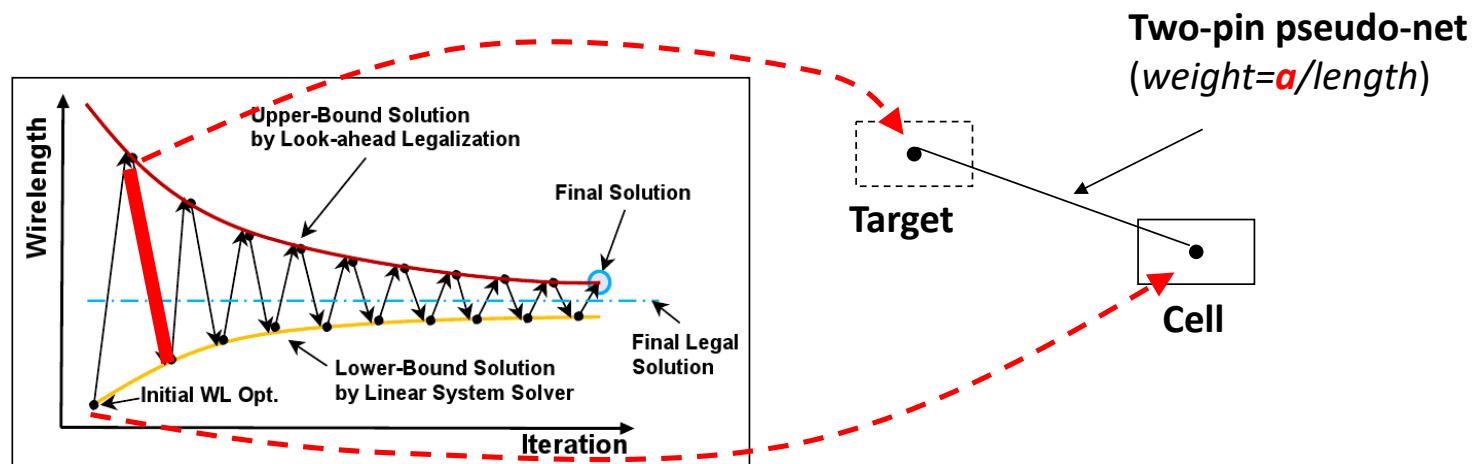
Lower-Upper-Bound Framework

- Lower-bound: minimize HPWL
- Upper-bound: rough legalization, routability-driven cell spreading
- Pseudo-net in lower-bound: Force cell move nearby the target position of upper-bound

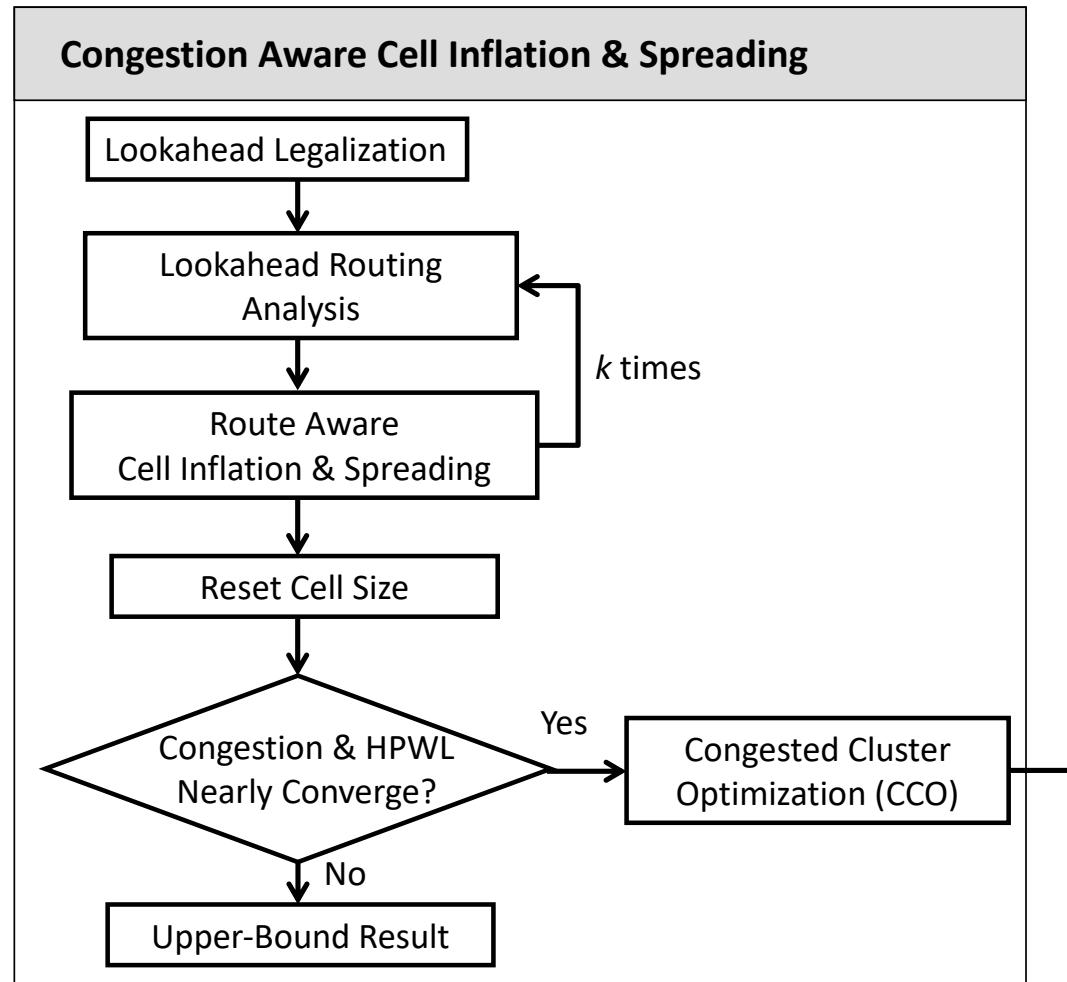


Lower-Bound Wirelength Optimization

- Pseudo-net
 - Consider cell coordinate obtained in the upper-bound computation
 - The weight “ a ” for pseudo-net is increasing during iterations

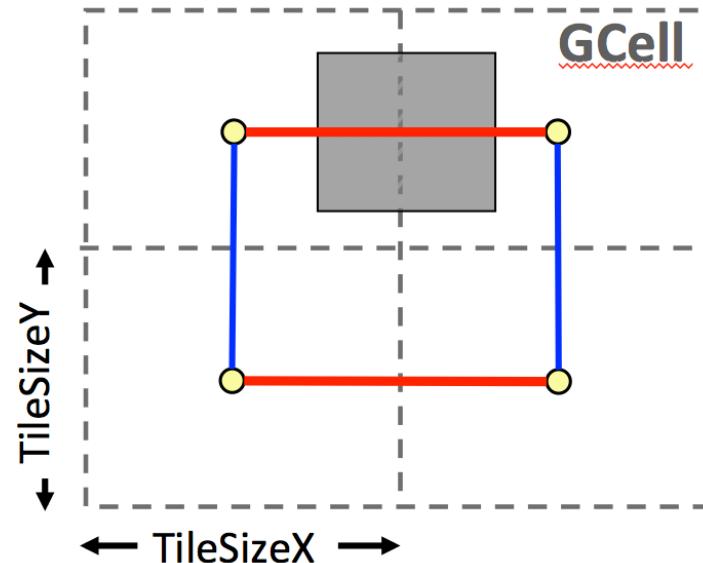


Upper-Bound Congestion Aware Optimization



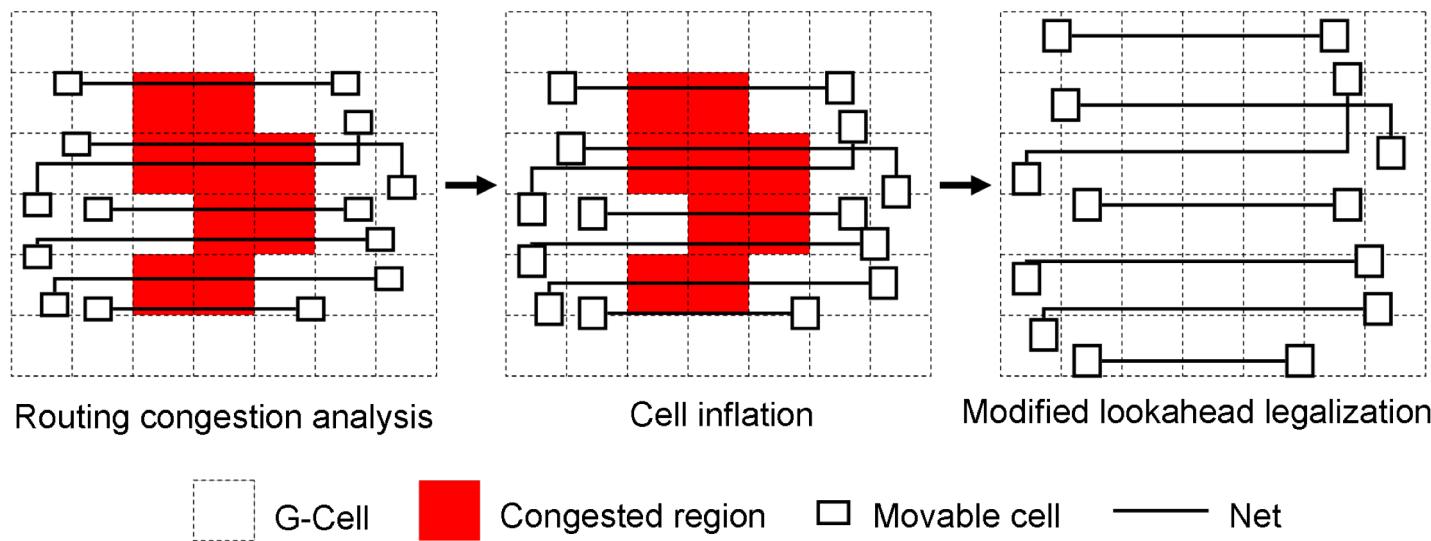
Lookahead Routing Analysis

- The chip is divided into uniform GCells.
- Supply of GCell edge
 - Total available routing tracks of all metal layers
- Demand of GCell edge
 - Number of nets passing through GCell edge
 - Number of pins in GCell



Cell Inflation & Spreading

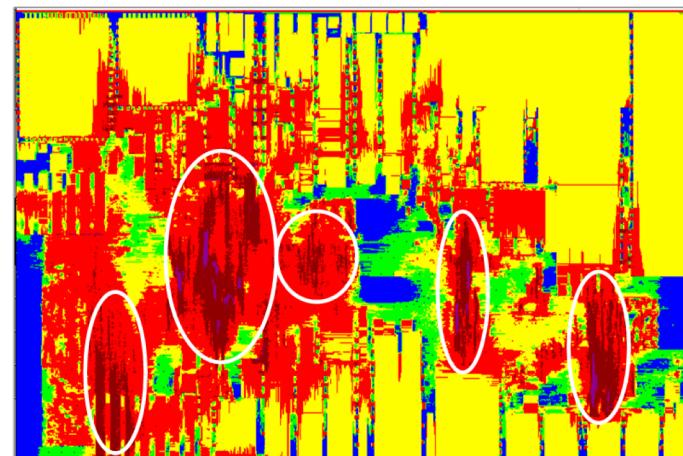
- Routing congestion reduction in horizontal direction



- Relieve congestion from local, semi-global and more importantly global nets

Congested Cluster Optimization (CCO)

- Congested clusters: A lot of interconnections
- The cells need to be spread more sparsely



$\text{Max}\{\text{DemandH} + \text{BlockTrackH}/ \text{DefaultSupplyH},$
 $(\text{DemandV} + \text{BlockTrackV})/ \text{DefaultSupplyV}\} :$

| | |
|-----------|-------------|
| 0% ~ 40% | 80% ~ 100% |
| 40% ~ 60% | 100% ~ 110% |
| 60% ~ 80% | >110% |

Routing result by NCTUgr (superblue3)

Congested Cluster Optimization (CCO)

- Congestion and HPWL between lower and upper bounds nearly converge
- Identification of congested clusters
- Full process of global router (NCTUgr³)
- These congested regions usually contain many interconnections

³ W. Liu, W. Kao, Y. Li, and K. Chao, “Multi-threaded collision-aware global routing with bounded-length maze routing”, in DAC, pp. 200-205, ACM, 2010

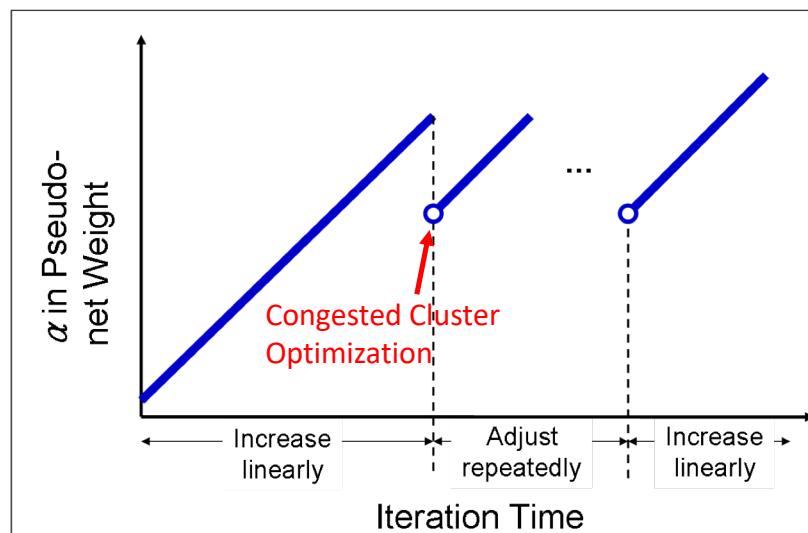
Congested Cluster Optimization (CCO)

- Enlarge cell size

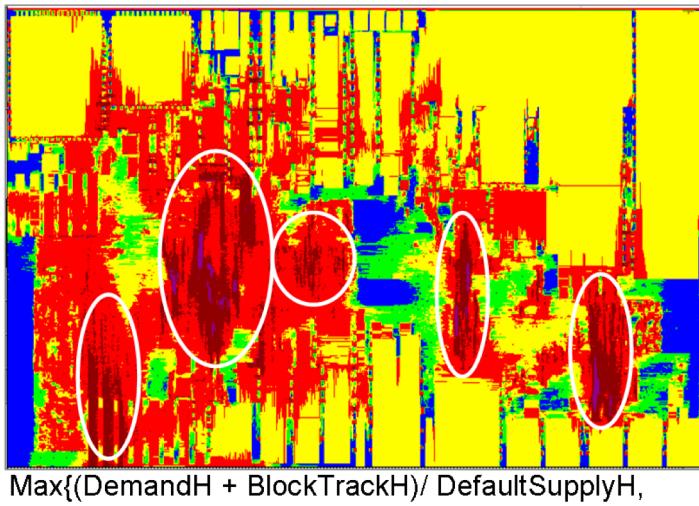
$$H_i = H_i \times \frac{DemandH_{e_1} + DemandH_{e_2}}{SupplyH_{e_1} + SupplyH_{e_2}}$$

- Adjustment of pseudo-net weight

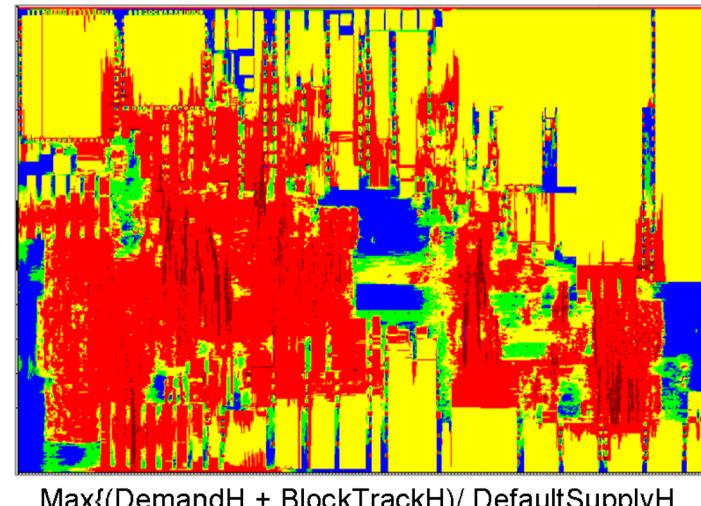
$$pseudo_i = \alpha / (|lx_i - ux_i| + |ly_i - uy_i|)$$



Congested Cluster Optimization (CCO)



(a)

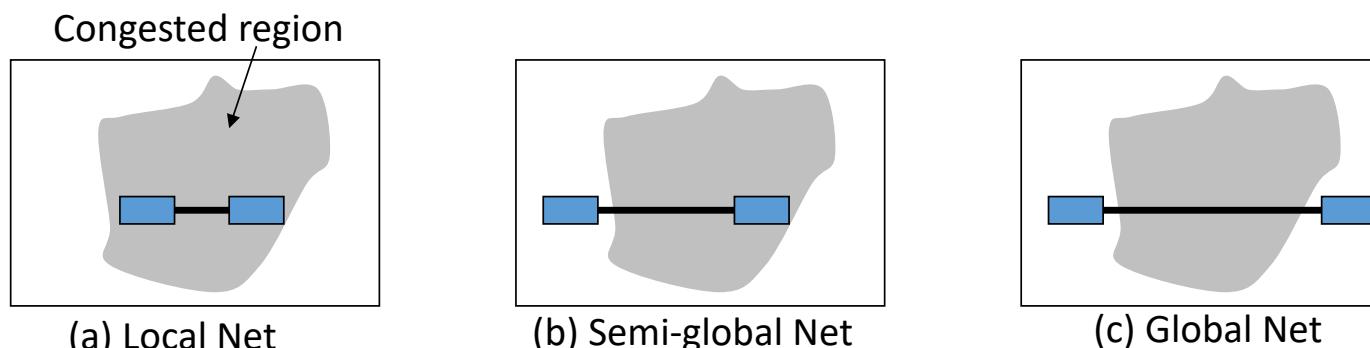


(b)

(a) Previous routing result (b) Using congestion cluster optimization (superblue3)

Post-Processing

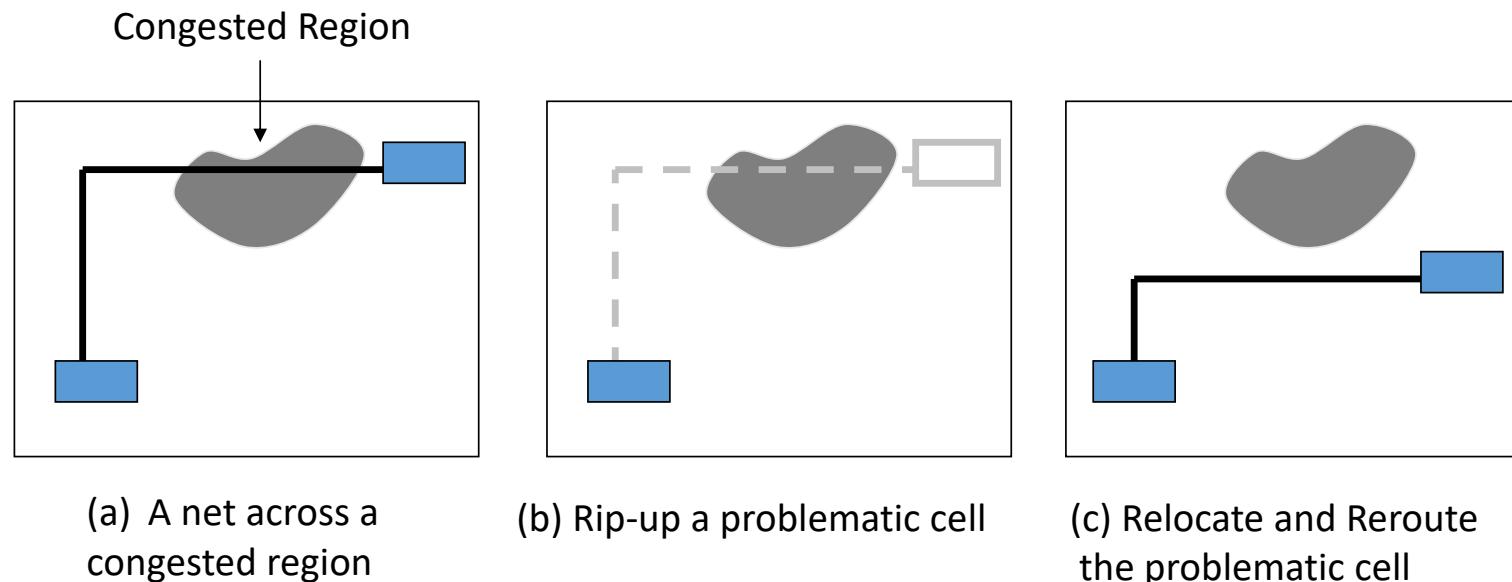
- Input: placement result; corresponding routing result
- Overflow (routing congestion) is from:
 - Local nets
 - Semi-global nets
 - Global nets



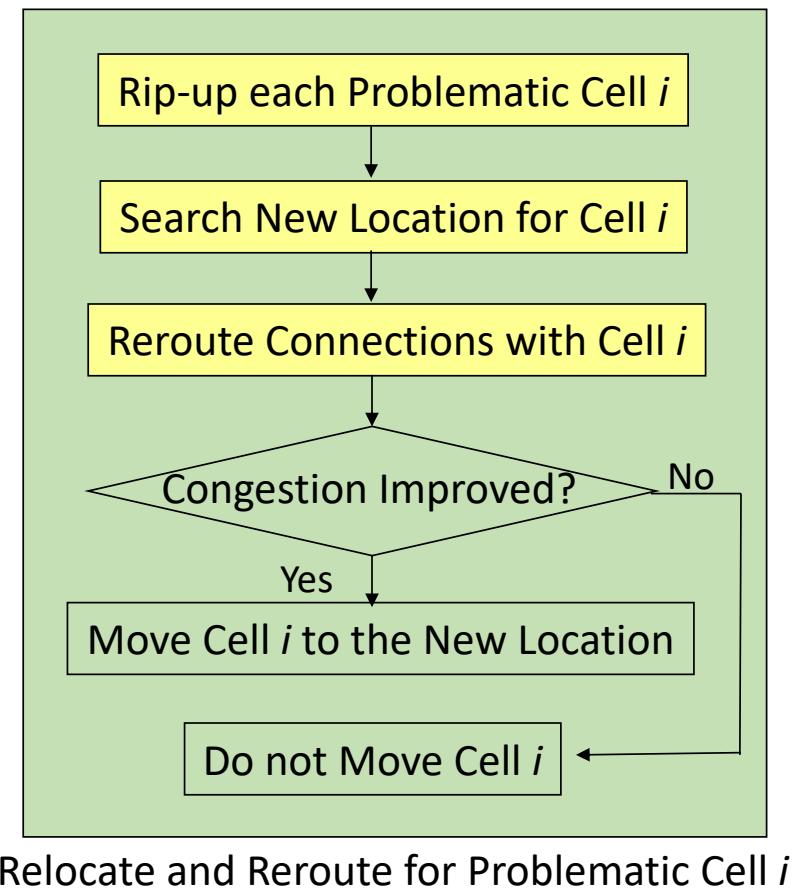
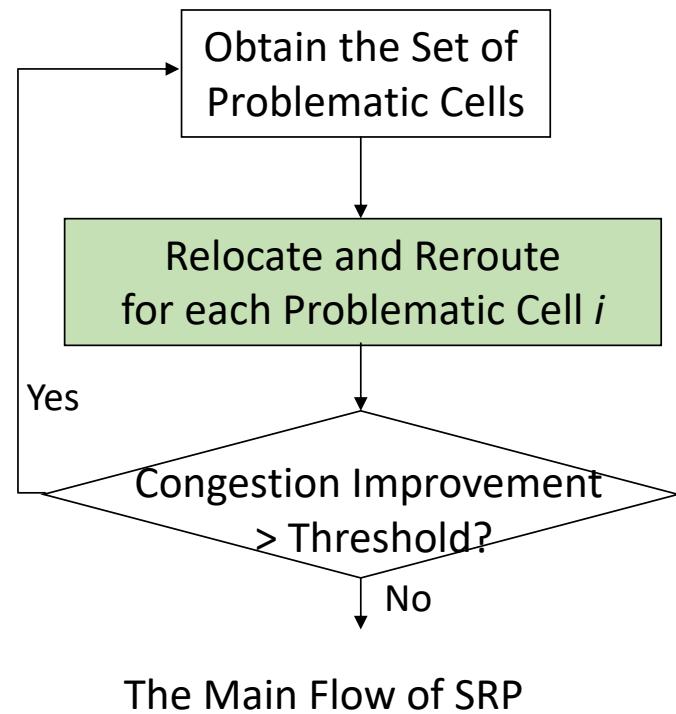
- Most of the previous works directly solve the congestion from **local nets only**.

Why SRP (Simultaneous Route and Place)?

- An example of SRP

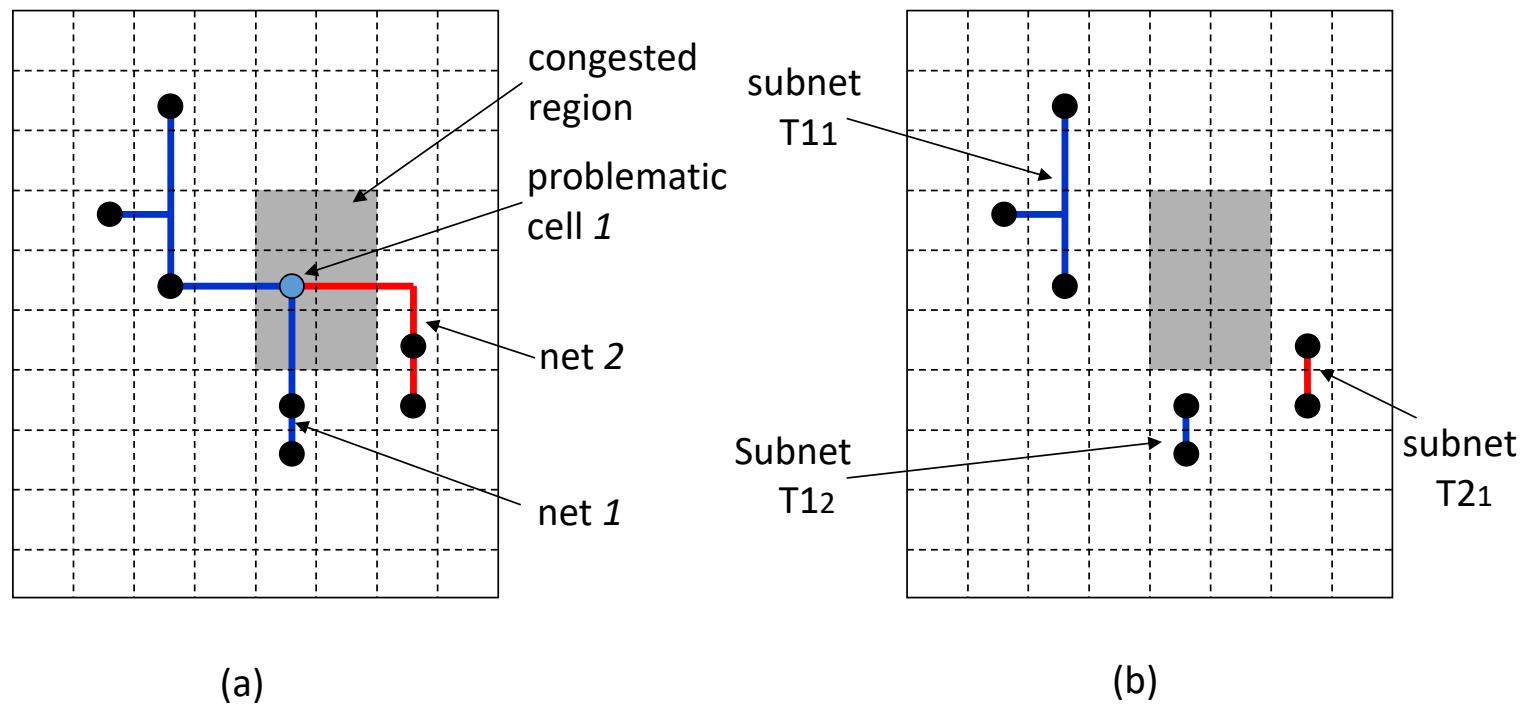


Overview of SRP



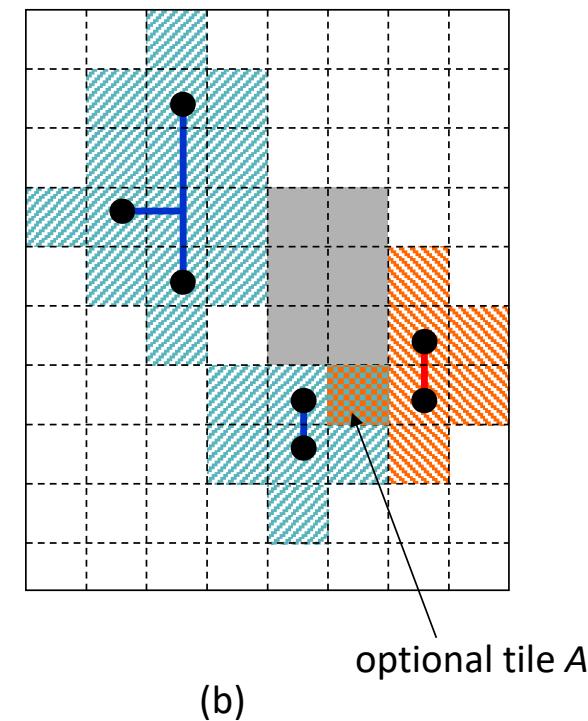
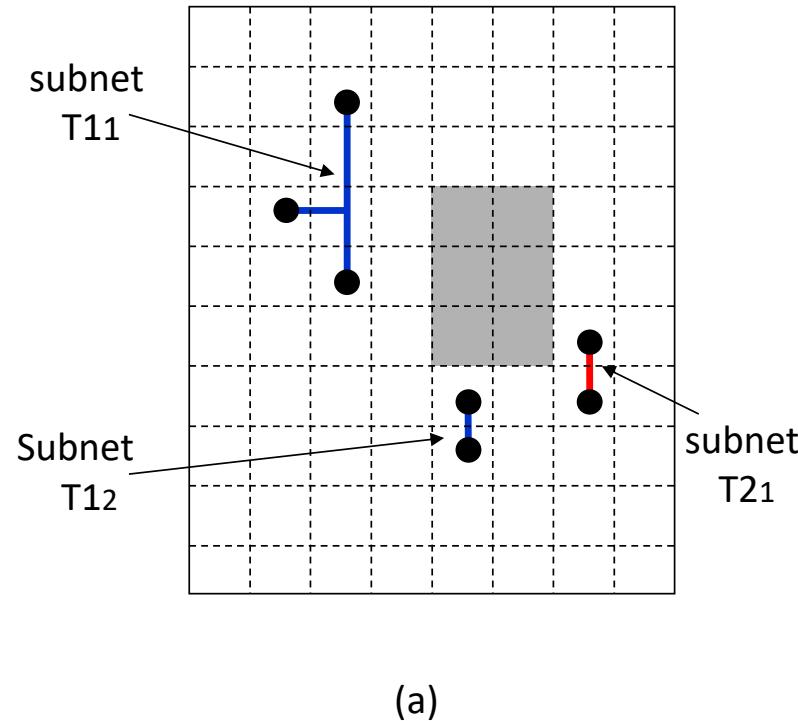
Rip-up Problematic Cell

- Besides removing cell, its connections of each associated nets have to be removed as well.



Search New Location

- Multi-source Propagation



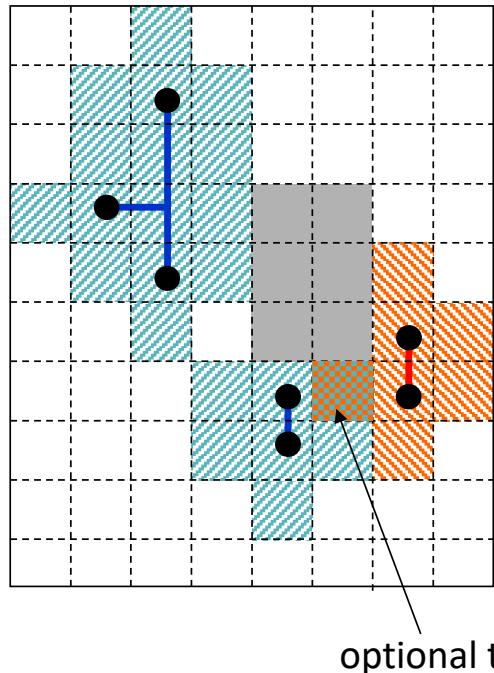
Reroute New Location

- For each associated net i , connect its subnets¹ together by multi-subnet maze routing
 - While the number of subnets >1
 - Propagate from subnet 1
 - Find a shortest path between subnet 1 and any other subnet j
 - $\text{subnet } 1 = \text{subnet } 1 \cup \text{subnet } j$

¹The new location of the current problem cell is seen as a subnet as well.

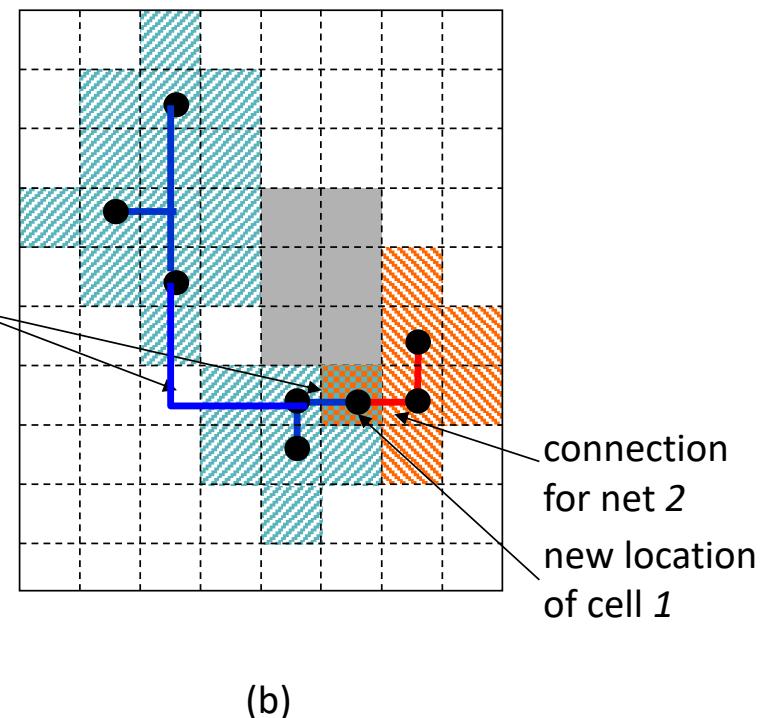
Reroute New Location

- An example



(a)

connections
for net 1



(b)

connection
for net 2
new location
of cell 1

ePlace

Global Placement

- $W(x, y) = \sum_n \max_{i \in n} x_i - \min_{i \in n} x_i + \max_{i \in n} y_i - \min_{i \in n} y_i$
- Constraint: *Demand b < Capacity b* \forall bin b .

ePlace

- *Function:* $W(x, y) + \lambda D(x, y)$,
 - $D(x, y)$ total charge potential

Shadow Price of Primal Dual Formulation

- *Function:* $W(x, y) + \sum_b \mathcal{V}_b D_b(x, y)$,
 - $D_b(x, y)$ charge potential in bin b
- The shadow price of bin i is proportional to the demand/capacity constraint violation.

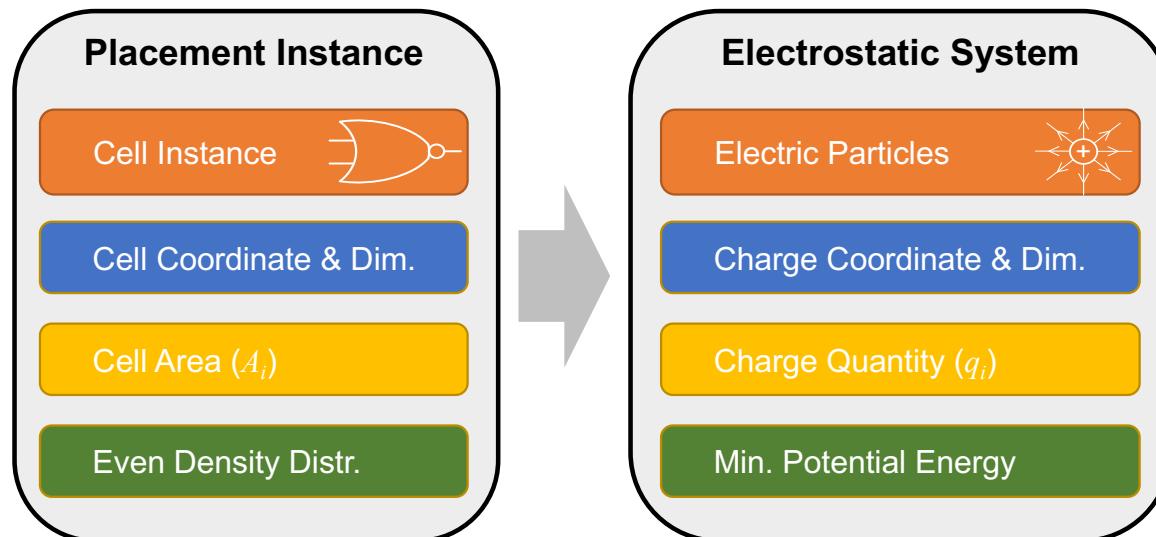
Problem Formulation

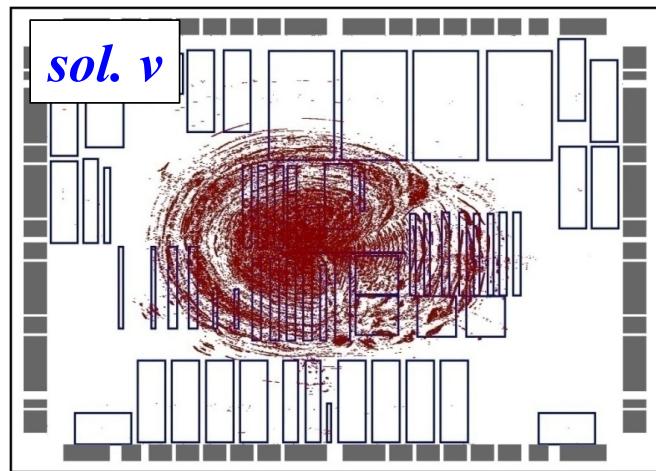
這邊的重點就是設計function的方式

- Placement Objective Function $f(v)$

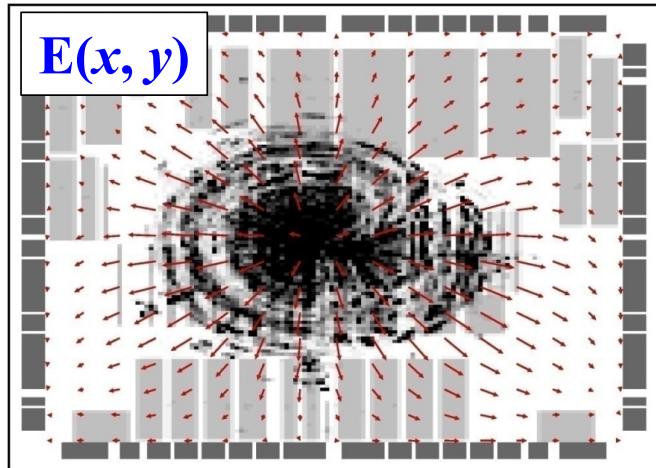
$$\min_v f(x, y) = W(x, y) + \sum_b \mathcal{V}_b D_b(x, y)$$

- ePlace: Electrostatics-based global-smooth density function

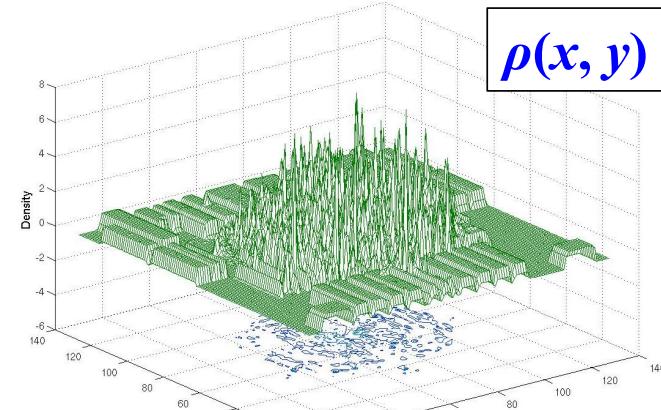




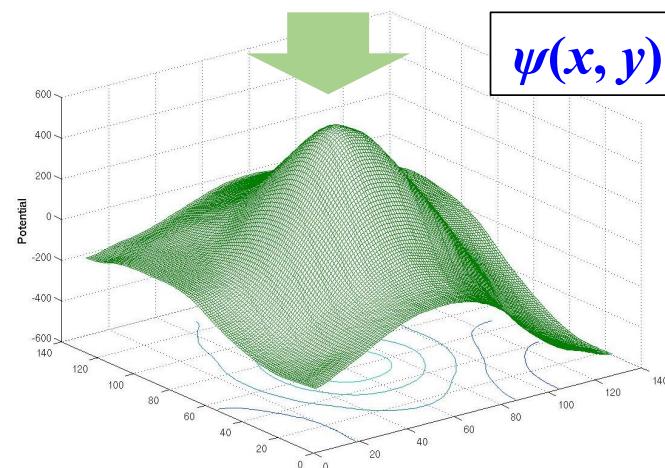
cell & macro distr.



electric field distr.

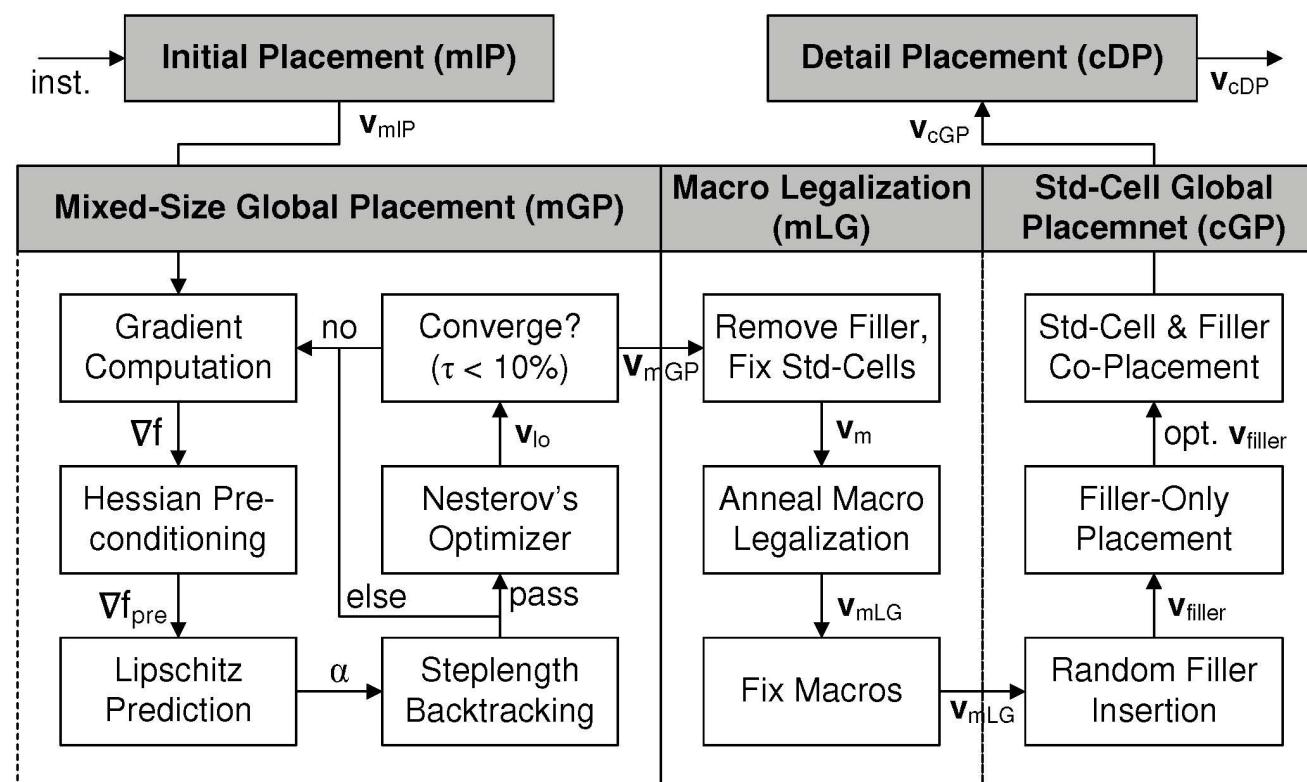


charge density distr.



electric potential distr.

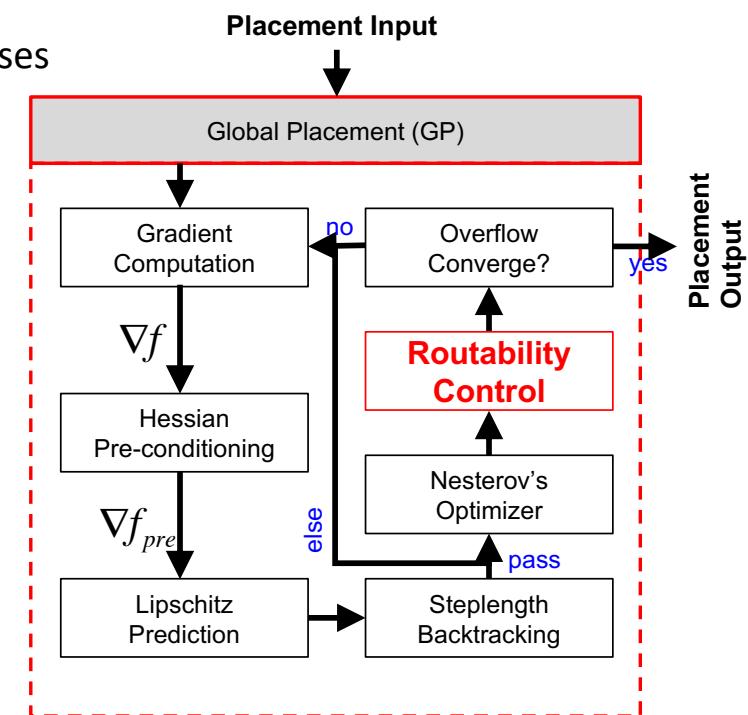
Flow of ePlace



ePlace 2.0: Routability-Driven Placement

- **Cell Inflation**

- Each cell inflation procedure: (6-10) times of inflation phases
- (4-6) times of inflation procedures
- Horizontal(vertical) congestion → inflate in V(H) direction
- $routeOVFL_h(g_i) = \frac{TrackDemand_v(g_i)}{TrackSupply_v(g_i)}$
- Implementation
 - Alternate H and V inflation
 - Continuous inflation phase
 - 6-10 times before restoring the cell size
 - Inflation Bound
 - total inflation area ≤ white space



[2] X. He, T. Huang, L. Xiao, H. Tian and E. F. Y. Young, "Ripple: A Robust and Effective Routability-Driven Placer", IEEE Trans. CAD, 32(10) (2013), pp. 1546-1556.

[3] J. Lu, P. Chen, C.-C. Chang, L. Sha, D. J.-H. Huang, C.-C. Teng and C.-K. Cheng, "ePlace: Electrostatics Based Placement Using Nesterov's Method", Proc. DAC, 2014, pp. 1-6.

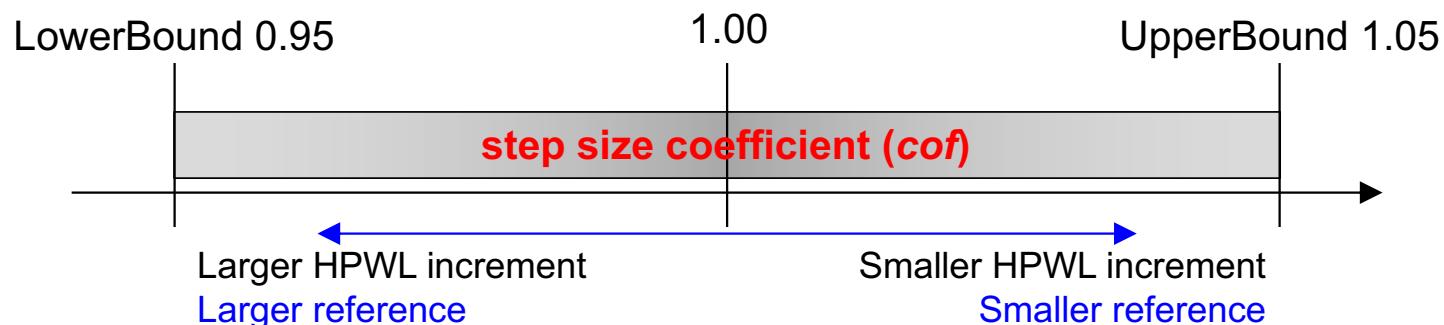
ePlace 2.0: Dynamic Step Size

- Dynamic Step Size in the Previous ePlace

- Accounts for $\text{diff}(\text{HPWL})$.

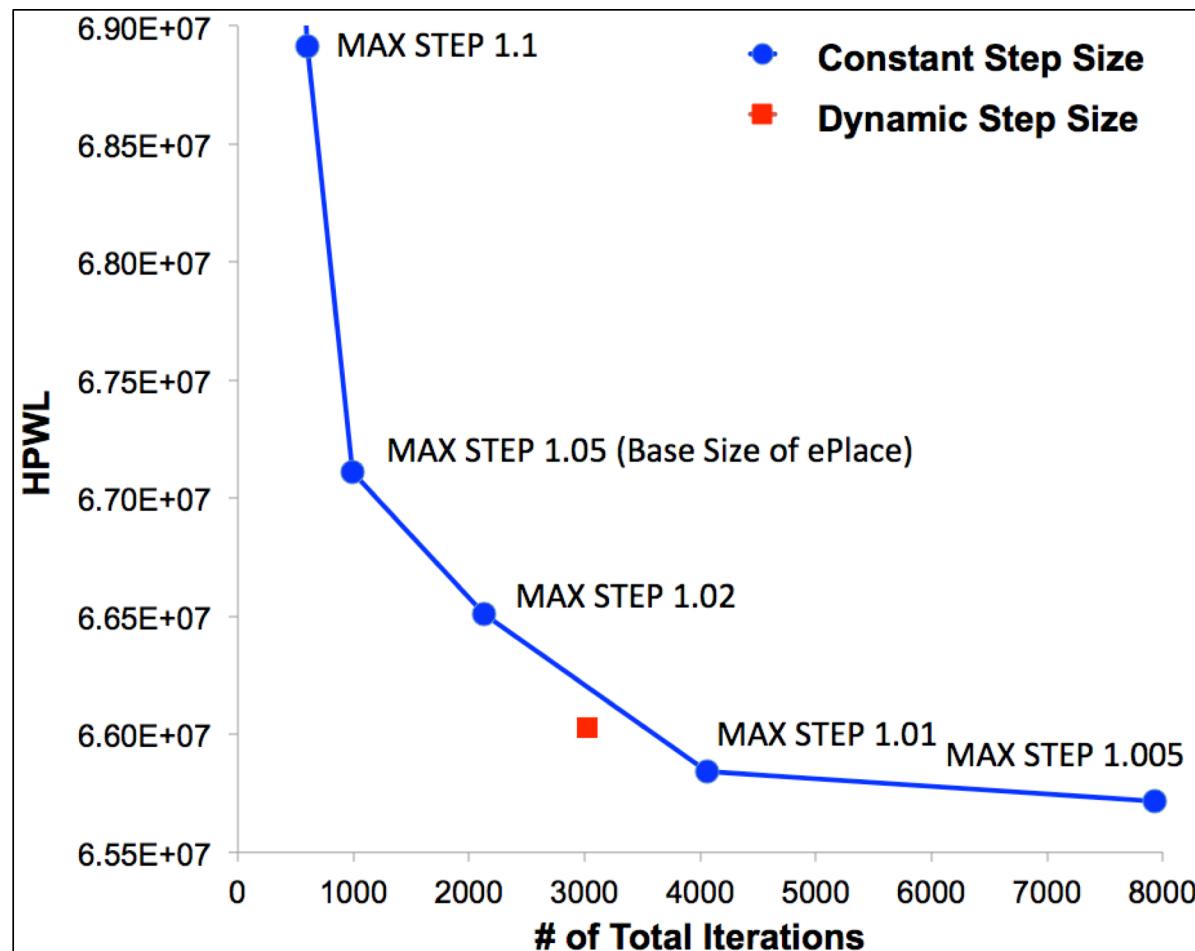
$$(HPWL_{iter} - HPWL_{iter-1}) / reference_delta_wirelength$$

- For each iteration, get ‘reference’.
 - If **reference** < 0, then **step size coefficient** is the upper bound value.
 - If **reference** > 0, then



ePlace 2.0: Improved Dynamic Step Size

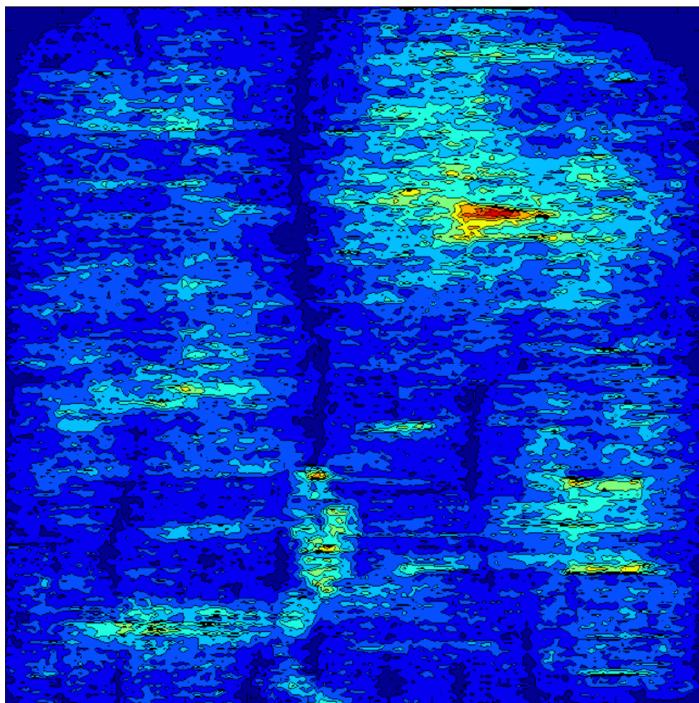
- Solution quality in terms of final HPWL
 - ADAPTEC1
 - CPUtime
 - 30min (1.05)
 - 60min (1.02)
 - 90min (DS)
 - 120min (1.01)



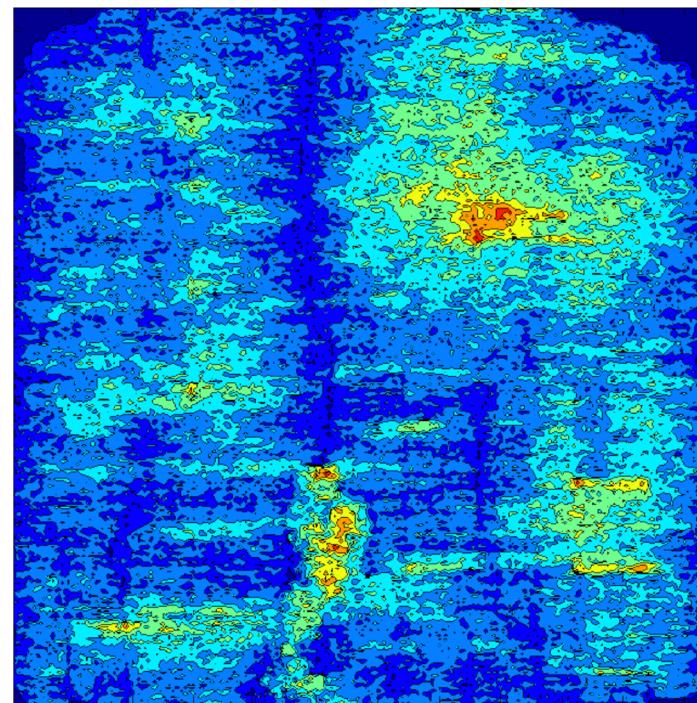
Source from Prof. C.K. Cheng of UCSD

ePlace 2.0: Routability-Driven Placement

- leon3mp (industrial testcase)
 - #STD cells: 437465, #I/O ports: 332, #nets: 437718, #pins: 1388822



Horizontal Congestion by ePlace 2.0

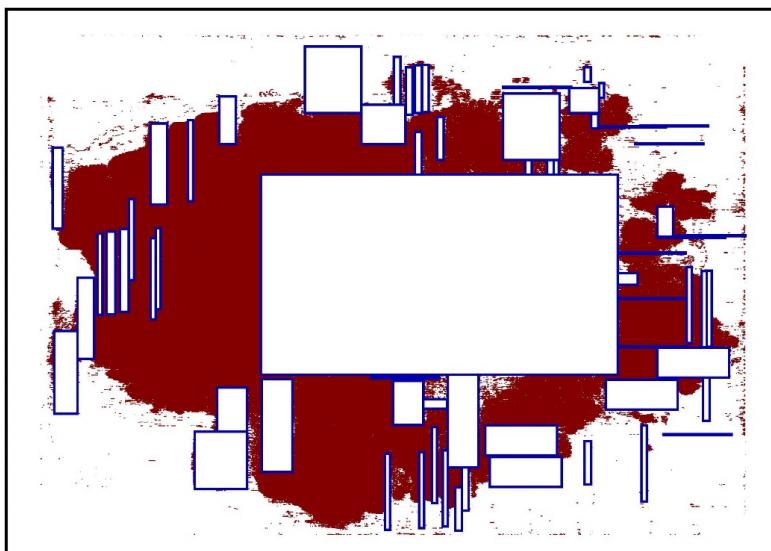


Horizontal Congestion by commercial tool

Source from Prof. C.K. Cheng of UCSD

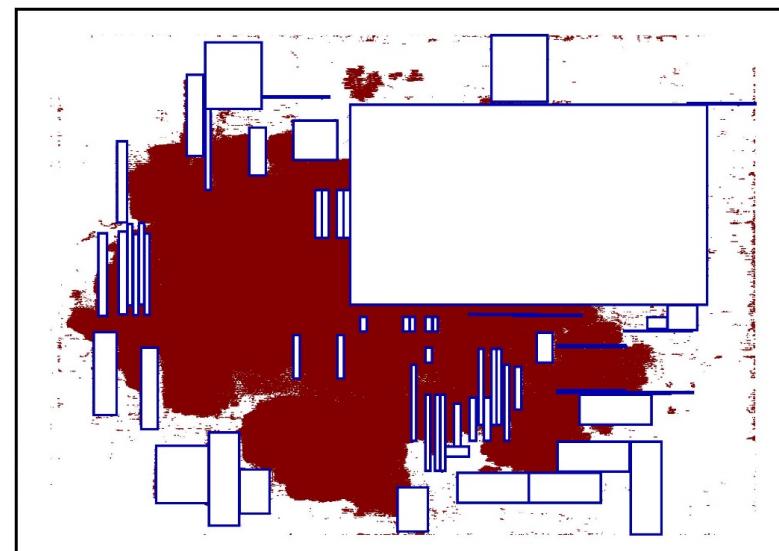
ePlace 2.0 Constraint-Driven Placement

Final placement results after detailed placement.



[ePlace]

- HPWL = 6.38E+7 (After Detailed Placement)
- #Iter = 1089 (615 + 474)
- runT = 43 min

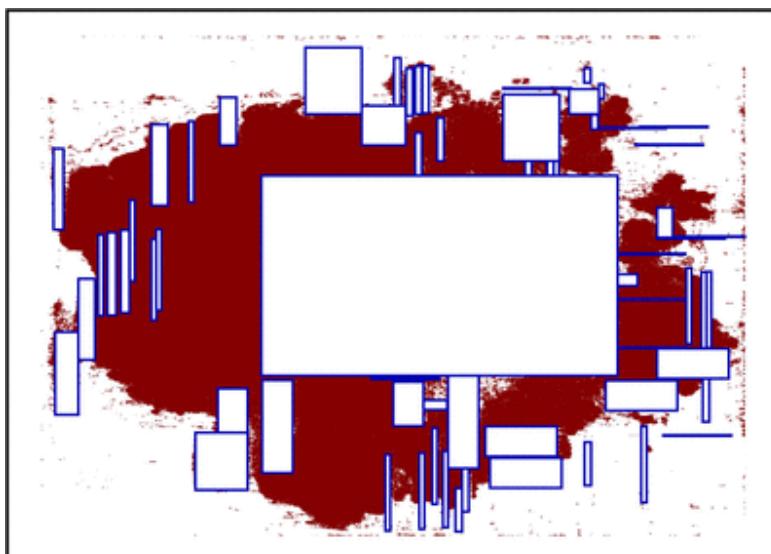


[ePlace 2.0 with Constraint-Driven]

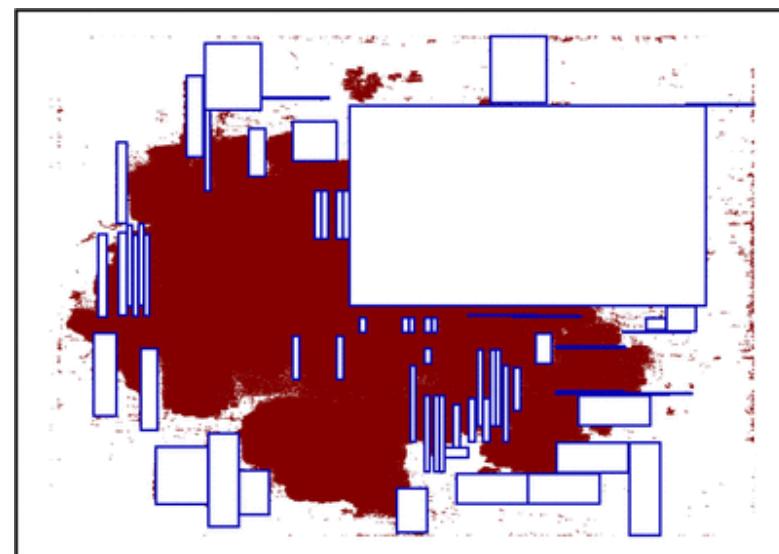
- HPWL = 5.71E+7 (After Detailed Placement)
- #Iter = 1078 (609 + 469)
- runT = 42 min

ePlace 2.0 Constraint-Driven Placement

Cell movement/location per each iteration



[ePlace]



[ePlace 2.0 with Constraint-Driven]

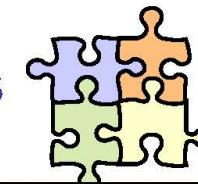
- HPWL = 6.38E+7 (After Detailed Placement)
- #Iter = 1089 (615 + 474)
- runT = 43 min

- HPWL = 5.71E+7 (After Detailed Placement)
- #Iter = 1078 (609 + 469)
- runT = 42 min

RePIAce

- RePIAce Overview
 - A single placement engine which solves five classes of benchmarks
 - Local Lagrange multiplier
 - Enables local smoothing that comprehend local over-demanded bins
 - Meta parameter tuning
 - Adjust step size of numerical method
 - Routability optimization
 - Simple but effective metal layer-aware superlinear cell inflation technique
- Differences from the previous *ePlace 2.0*
 - More experiments using DAC-2012 and ICCAD-2012 global routability-driven placement benchmark suites with new routability optimization techniques, significantly improved.
 - Code optimizations have significantly improved runtime for RePIAce ($\approx 4x$ faster)
 - Misc. improvements contribute to improve the existing placement mechanism
 - E.g., macro legalization using annealing, the amount of rollback after fixing macro, pushing more overflow, bin size determination and local smoothing, taking-off z-dimension computations, etc.

Modern Academic Analytical Placers



| Placer | Wirelength Model | Overlap Reduction | Integration | Optimization |
|--------------------|------------------|--------------------------|-------------------|-------------------------|
| APlace | LSE | Bin Density | Penalty Method | Nonlinear |
| BonnPlace | Quadratic | Partitioning | Region Constraint | Quadratic |
| DPlace | Quadratic | Diffusion | Fixed Point | Quadratic |
| ePlace | WA | Density (electrostatics) | Penalty Method | Nonlinear (Nesterov) |
| FastPlace | Quadratic | Cell Shifting | Fixed Point | Quadratic |
| FDP | Quadratic | Density | Fixed Point | Quadratic |
| Fuzhou | WA/LSE | Bin Density | Penalty Method | Nonlinear (subgradient) |
| Gordian | Quadratic | Partitioning | Region Constraint | Quadratic |
| hATP | Quadratic | Partitioning | Region Constraint | Quadratic |
| Kraftwerk2 | Bound2Bound | Bin Density | Fixed Point | Quadratic |
| mFAR | Quadratic | Density | Fixed Point | Quadratic |
| mPL6 | LSE | Bin Density | Penalty Method | Nonlinear |
| NTUplace3/4 | WA/LSE | Bin Density | Penalty Method | Nonlinear (gradient) |
| Ripple | Bound2Bound | Partitioning | Penalty Method | Quadratic |
| RQL | Quadratic | Cell Shifting | Fixed Point | Quadratic |
| SimPL | Bound2Bound | Partitioning | Penalty Method | Quadratic |
| Vassu | LSE | Assignment | Fixed Point | Nonlinear |

Source from Prof. Yaowen Chang of NTU @ Summer School 2018

Outline

- ASIC Placement and Basics
- FPGA Placement
- AI Engine Accelerated Placer
- Detailed Placement with Multi-row Cells

FPGA Placement

- ▶ FPGA: more efficient than CPU/GPU, more flexible than ASIC
- ▶ Challenges: large scale, complicated design rules, stringent time requirement

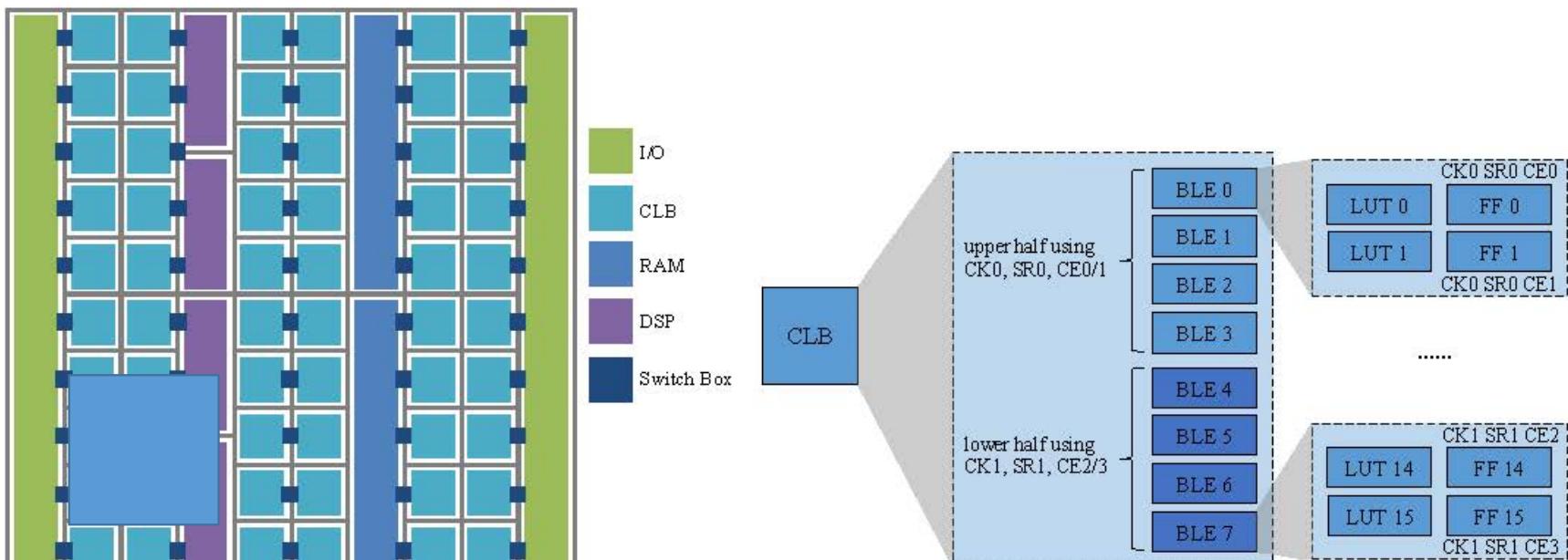
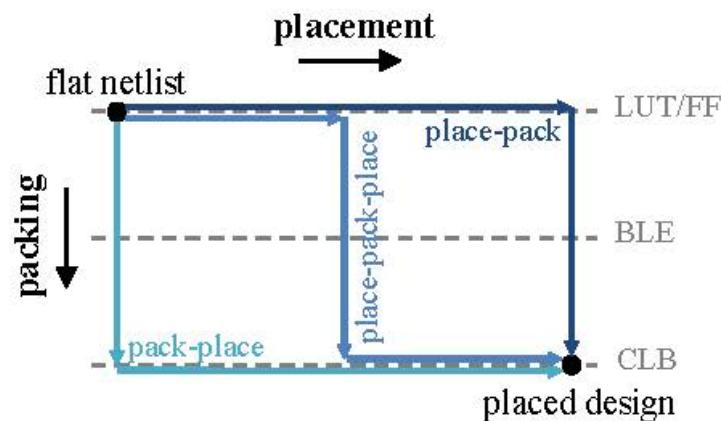


Illustration of Xilinx UltraScale architecture.

FPGA Placement

Three types of flows in previous work

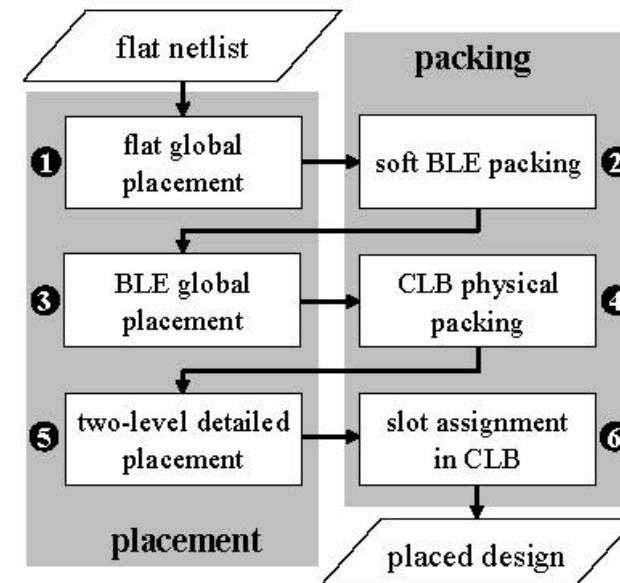
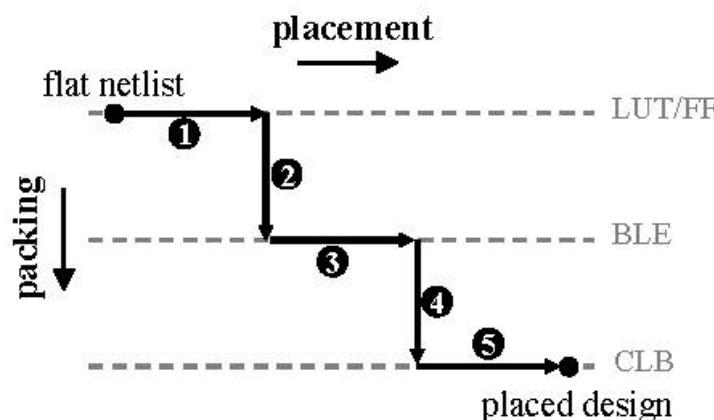
- ▶ pack-place (most traditional flow)
- ▶ place-pack
- ▶ place-pack-place



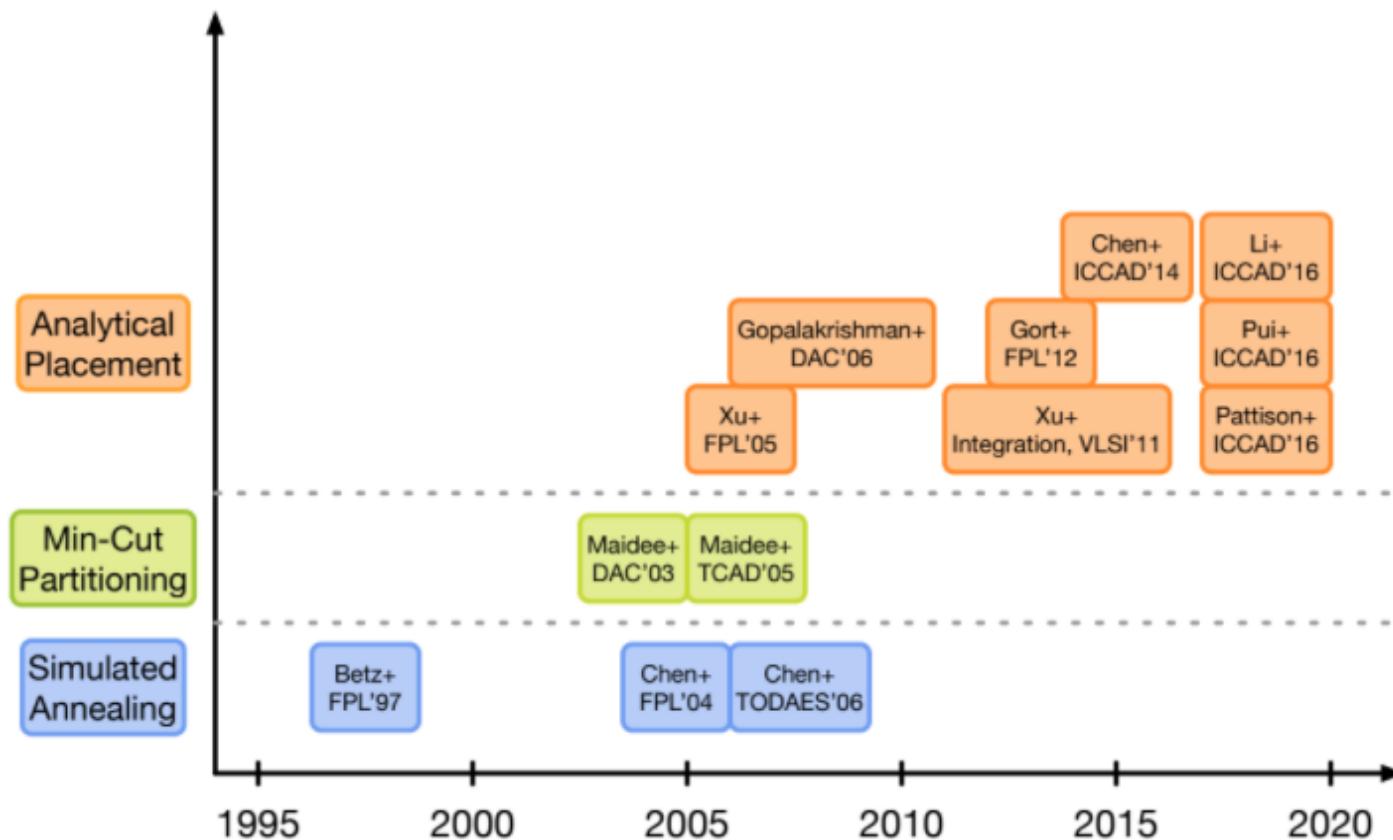
FPGA Placement

RippleFPGA

- ▶ Simultaneous packing and placement
- ▶ FPGA-routing-architecture-aware optimization

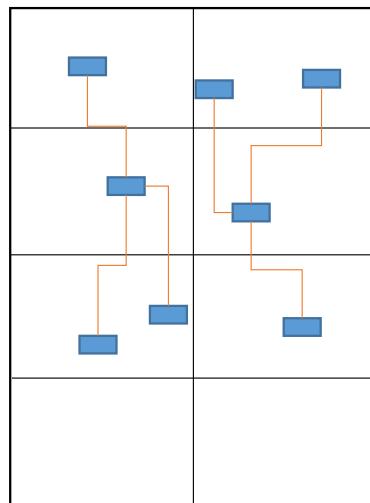


Previous Works

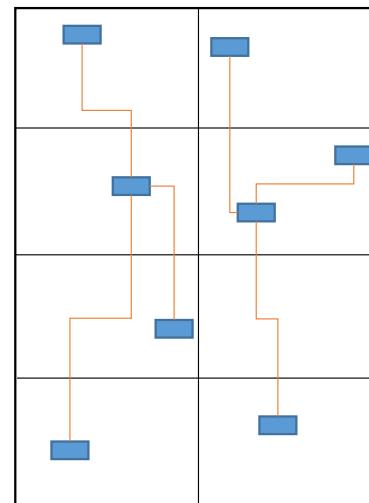


Partitioning

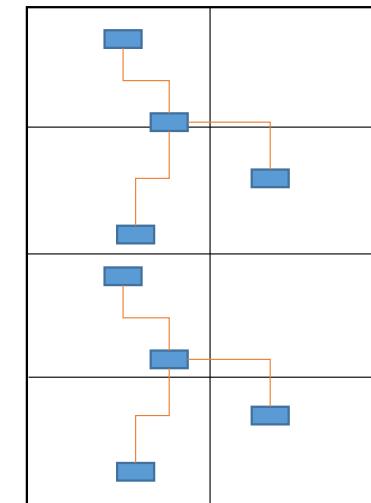
- Reduce congestion caused by **unbalanced** routing supply in the horizontal and vertical directions
- Cannot be resolved by spreading but by reallocating



(a)origin



(b)spreading

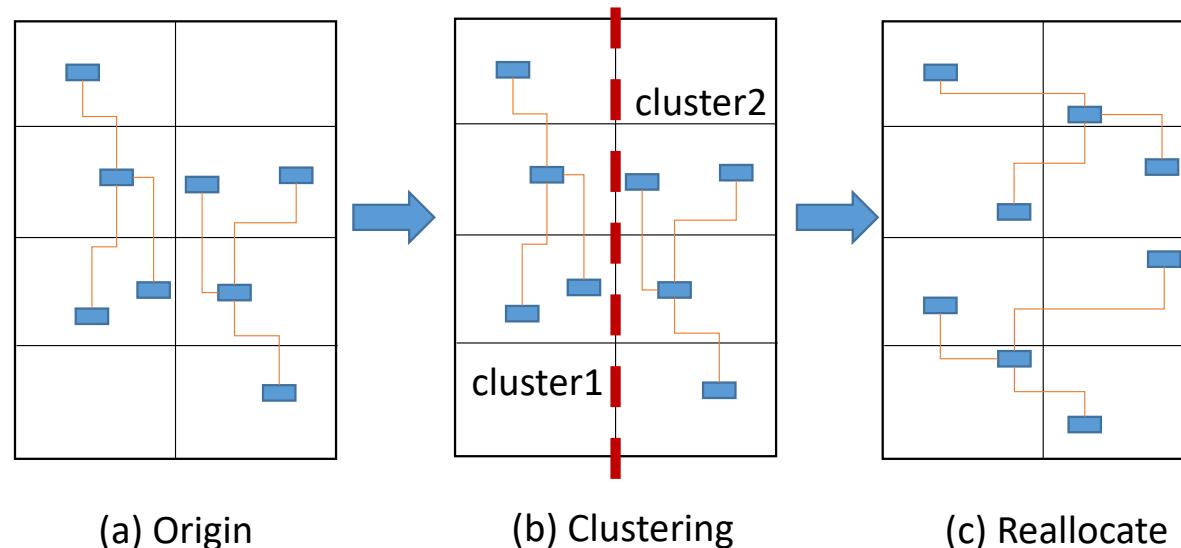


(c)reallocation

Comparison of different methods in solving congestion

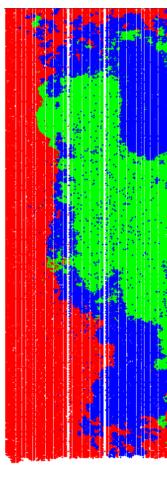
Partitioning

- Partition the circuit into sub-circuits using recursive bi-partitioning
- Reallocate the cells across the chip as sparse as we can
- Maintain relative order of clusters and cells inside the same cluster

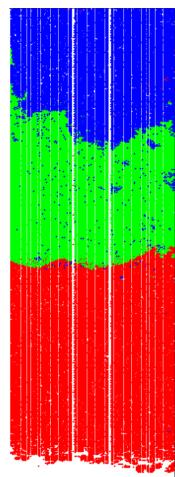


Partitioning

- Effect on real test case



(a) W/o partitioning

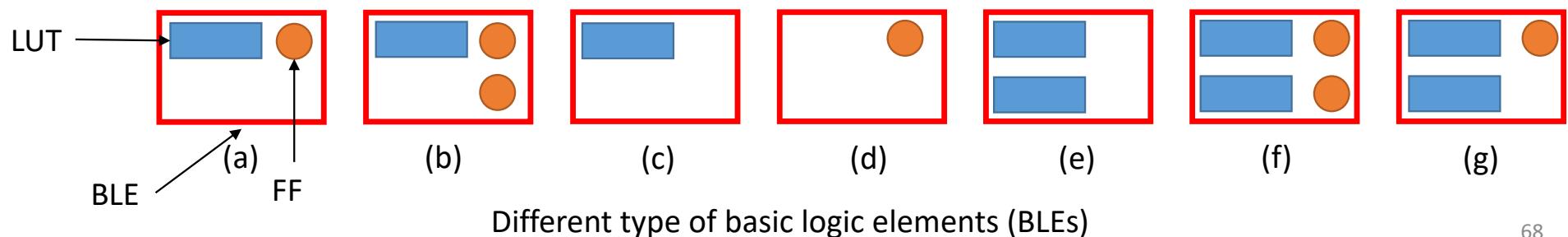


(b) With partitioning

| Design | baseline | | With partitioning | |
|---------|----------|---------------|-------------------|---------------|
| | WL | Normalized WL | WL | Normalized WL |
| FPGA-1 | 356314 | 1.000 | 353516 | 0.992 |
| FPGA-2 | 669133 | 1.000 | 649051 | 0.970 |
| FPGA-3 | 3532052 | 1.000 | 3277033 | 0.928 |
| FPGA-4 | 5556177 | 1.000 | 5508595 | 0.991 |
| FPGA-5 | - | - | - | - |
| FPGA-6 | 6549187 | 1.000 | 6238933 | 0.953 |
| FPGA-7 | 9723248 | 1.000 | 9500233 | 0.977 |
| FPGA-8 | 8423217 | 1.000 | 8122288 | 0.964 |
| FPGA-9 | 12050941 | 1.000 | 12044246 | 0.999 |
| FPGA-10 | 7820378 | 1.000 | 7308750 | 0.935 |
| FPGA-11 | 11172550 | 1.000 | 10672421 | 0.955 |
| FPGA-12 | 8464954 | 1.000 | - | - |

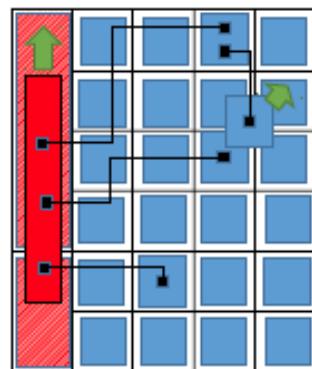
Packing

- LUTs and FFs are packed into basic logic elements (BLEs) to reduce the **inter-connections** between sites in routing:
 1. Forming basic logic elements(BLEs) that consist of only one LUT and at least one FFs
 2. Let the remaining LUTs and FFs be BLEs of itself only
 3. Merging two BLEs into one if their LUTs have many connections
 4. Maximum matching based (connection and distance as weight)

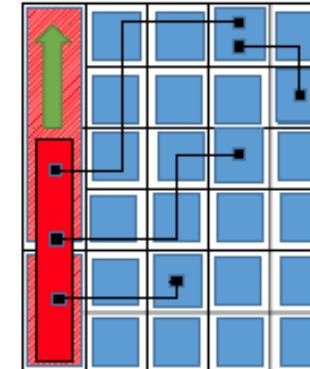


Global Placement Engine

- Quadratic placement based on Ripple
- Congestion-driven global placement
 - Inflate and shrink cell in some iteration to improve routability
 - Machine learning-based congestion estimation
 - Legalize DSP/RAM in the middle first using bipartite matching



Same displacement, difference in HPWL



Large displacement

Comparisons

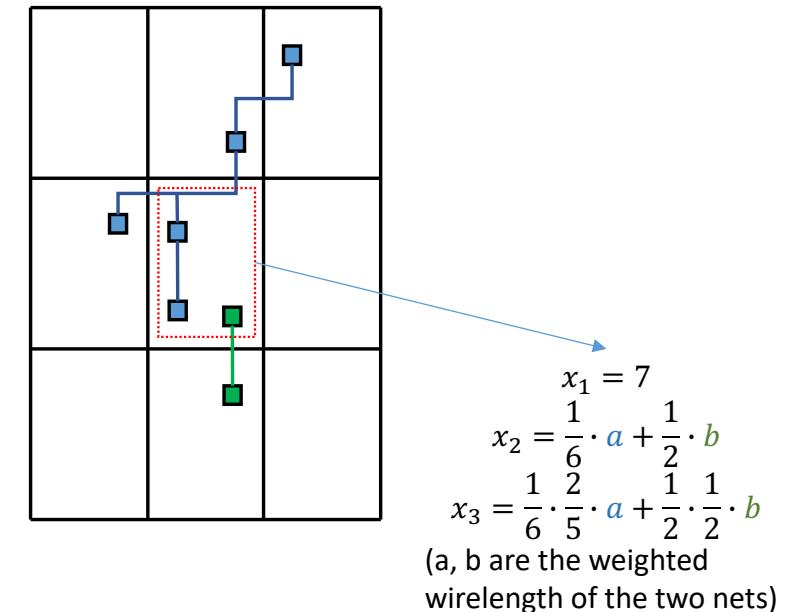
| Design | baseline | | With partitioning | | With congestion-driven GP | | With Both | |
|---------|----------|---------------|-------------------|---------------|---------------------------|---------------|-----------|---------------|
| | WL | Normalized WL | WL | Normalized WL | WL | Normalized WL | WL | Normalized WL |
| FPGA-1 | 356314 | 1 | 353516 | 0.992 | 356081 | 0.999 | 352628 | 0.990 |
| FPGA-2 | 669133 | 1 | 649051 | 0.970 | 655740 | 0.980 | 645400 | 0.965 |
| FPGA-3 | 3532052 | 1 | 3277033 | 0.928 | 3533351 | 1.000 | 3262106 | 0.924 |
| FPGA-4 | 5556177 | 1 | 5508595 | 0.991 | 5531058 | 0.995 | 5509661 | 0.992 |
| FPGA-5 | - | - | - | - | 10120061 | - | 9968955 | - |
| FPGA-6 | 6549187 | 1 | 6238933 | 0.953 | 6521672 | 0.996 | 6180104 | 0.944 |
| FPGA-7 | 9723248 | 1 | 9500233 | 0.977 | 9933324 | 1.022 | 9639639 | 0.991 |
| FPGA-8 | 8423217 | 1 | 8122288 | 0.964 | 8409769 | 0.998 | 8156951 | 0.968 |
| FPGA-9 | 12050941 | 1 | 12044246 | 0.999 | 12104565 | 1.004 | 12305192 | 1.021 |
| FPGA-10 | 7820378 | 1 | 7308750 | 0.935 | 7682063 | 0.982 | 7139694 | 0.913 |
| FPGA-11 | 11172550 | 1 | 10672421 | 0.955 | 11528235 | 1.032 | 11022815 | 0.987 |
| FPGA-12 | 8464954 | 1 | - | - | 8105645 | 0.958 | 7363451 | 0.870 |

ML-based Congestion Estimation

- Motivation:
 - More **accurate** and **less parameter tunings**
 - Previously used congestion estimation methods in FPGAs
 - Global routers for ASICs
 - Probabilistic models
 - Limitations:
 - Not tailored for FPGAs
 - A lot of parameters to set
- Goals of our methods
 - Try to mimic the behavior of congestion estimation of design tools from the device company
 - Assume the congestion estimation from the tool can guide the placement well
 - Study how to leverage machine learning to build a congestion model on FPGA

ML-based Congestion Estimation

- Congestion Model
 - G-Cells based, each corresponds to a switchbox
- Three Features for each G-Cell
 - Total number of pins of the net covering it
 - $x_1 = \sum_{m \in N_i} \#pins \text{ of net } m$
 - A weighted sum of BB box covering it
 - $x_2 = \sum_{m \in N_i} \frac{w_m \cdot HPWL_m}{\#gcell_m}$
 - Combining the two
 - $x_3 = \sum_{m \in N_i} \frac{p_{m,i}}{\#pins \text{ of net } m} \cdot \frac{w_m \cdot HPWL_m}{\#gcell_m}$



ML-based Congestion Estimation

- Learning Models
 - Global Linear Model
 - Consider the current site and its neighboring 8 sites
 - $y = f_{glm}(X) = \sum_{i=1}^{27} w_i x_i$
- Training Methods
 - Unified model
 - 70% training and 30% testing per design
- Result:
 - MPE (Mean Percentage Error): 11.5
 - r^2 : 0.943 (the higher the accuracy, the closer to 1.0)

ML-based Congestion Estimation

- Comparison
 - Global routers for ASICs
 - Cons: hard to set the routing capacity
 - Probabilistic models
 - Cons: only good correlation with the relative congestion level
 - Machine Learning-Based
 - Good correlation with the congestion level
 - Give a better sense of congestion level
 - Less parameter tuning

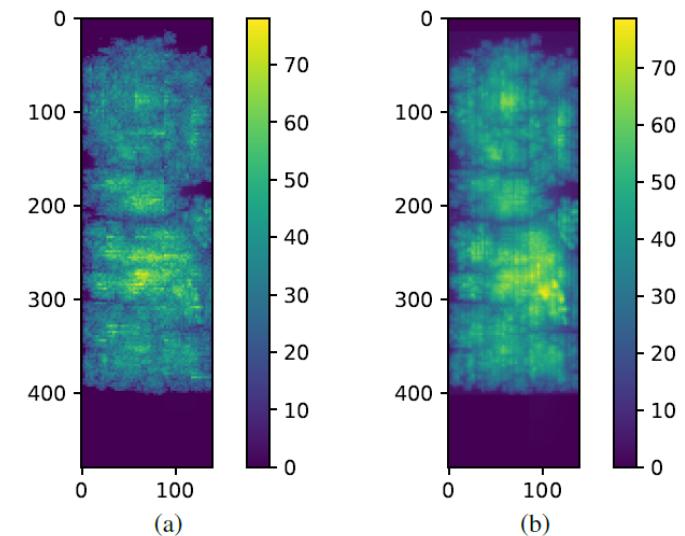
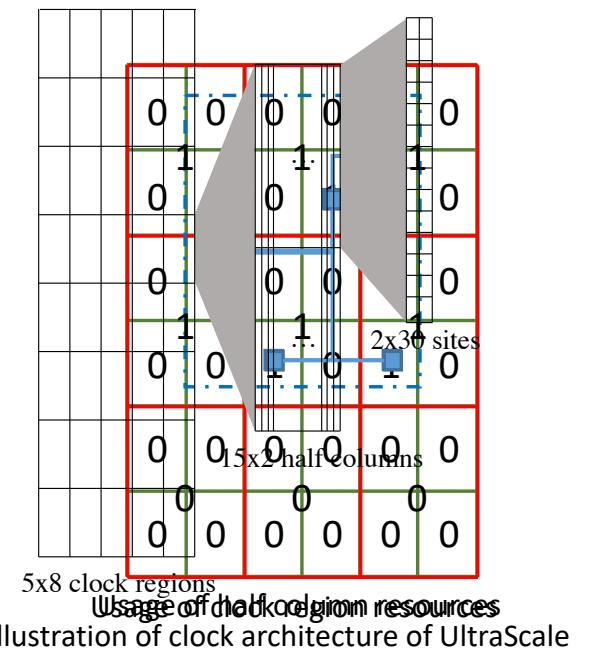


Fig. 7: Congestion Maps of CLK-FPGA13 in ISPD2017 contest: (a) routing congestion predicted by Vivado; (b) our estimation.

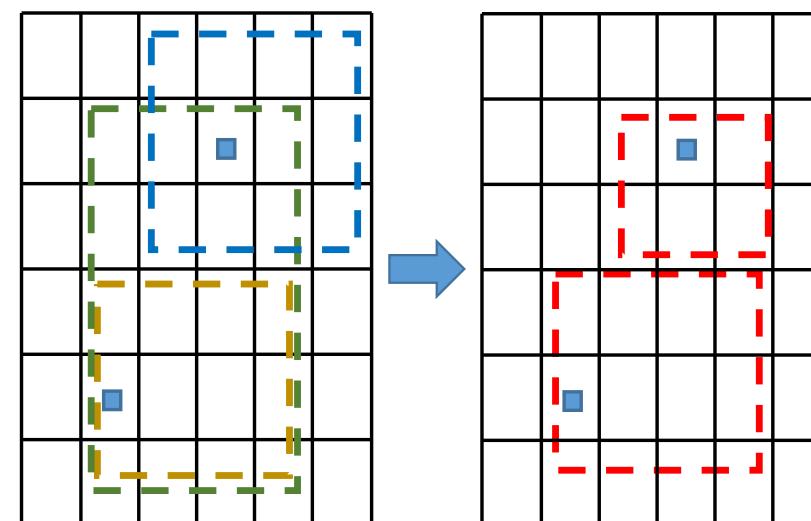
Clock Constraints Legalization

- Clock constraints of UltraScale FPGAs
 - Clock region constraints
 - Bound box of the clock net
 - Violation: #clock is larger than 32
 - Half column constraints
 - Loads of the clock net
 - Violation: #clock is larger than 16
- Displacement-driven two-step legalization
 - Clock region planning
 - Remove all the clock region violations after global placement
 - Half Column Legalization
 - Remove all the half column violations after legalization



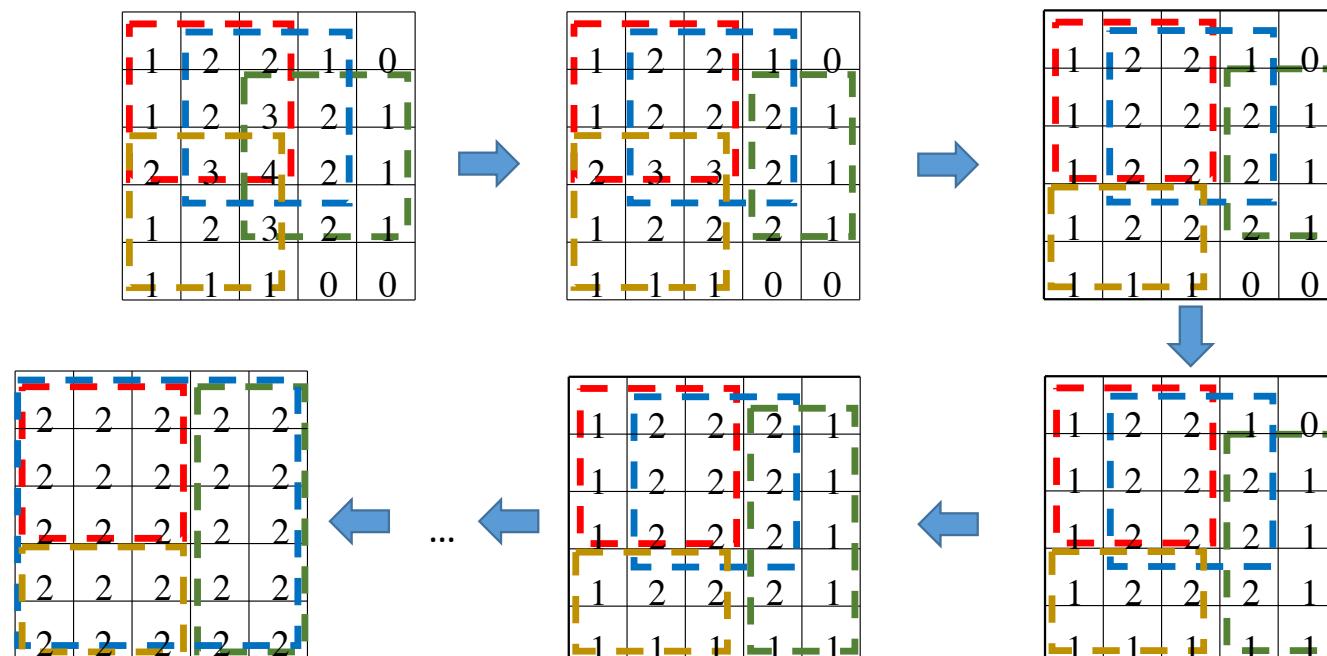
Clock Constraints Legalization

- Two-Stage Clock region planning
 - Assign a bounding box to each cell such that there will be no violation if they stay in the box
 - Shrink Stage
 - Expand Stage



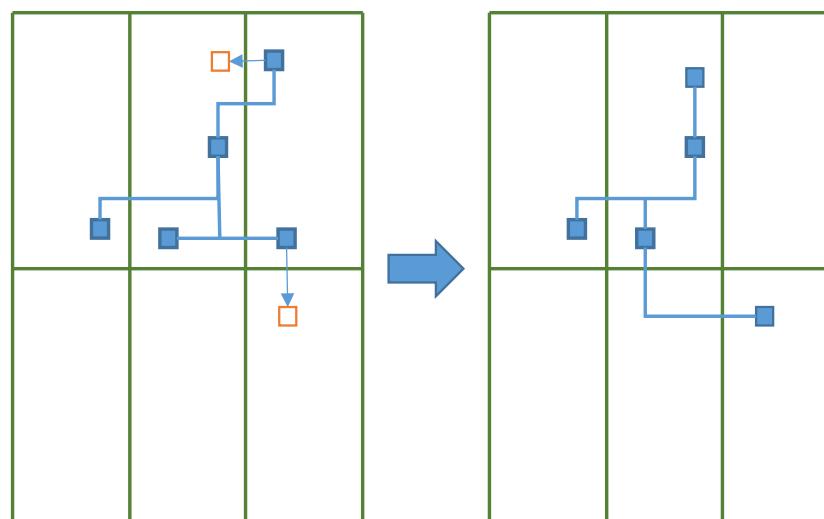
Clock Constraints Legalization

- Two-Stage Clock region planning
 - Shrink Stage
 - Expand Stage



Clock Constraints Legalization

- Half Column Legalization
 - All the future movement cannot induce any new half column violation
 - Iteratively select **the most overflow column** and remove the clock such that the **smallest displacement** is induced
 - Each load will be moved to its nearest site in another half column



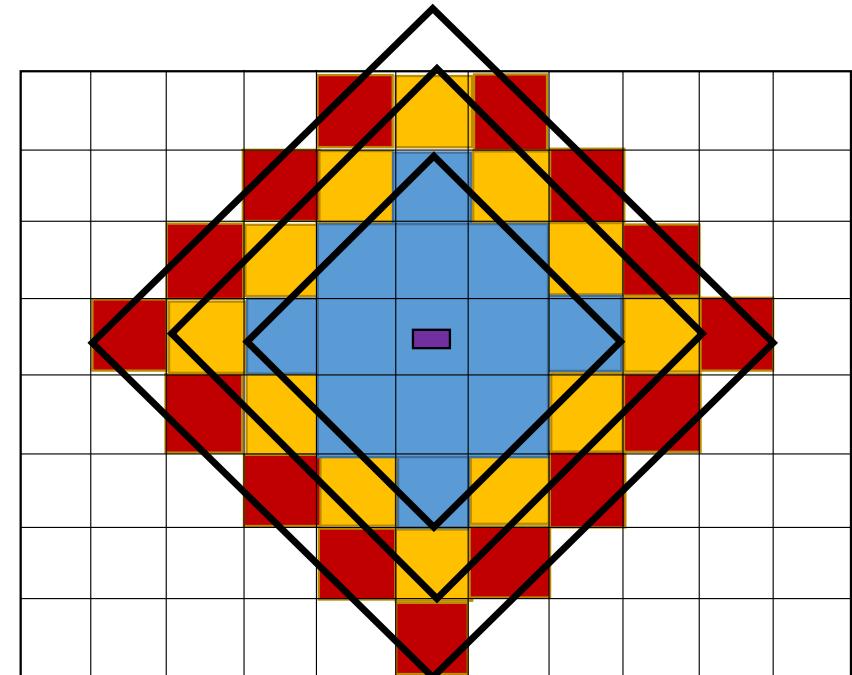
Comparisons

| Design | w/ CCL | | | | w/o CCL | | | |
|------------|---------|-------|------|-------|---------|--------------|------|--------------|
| | HPWL | ratio | Time | ratio | HPWL | ratio | Time | ratio |
| CLK-FPGA01 | 1582915 | 1 | 288 | 1 | 1582917 | 1.000 | 276 | 0.958 |
| CLK-FPGA02 | 1577051 | 1 | 266 | 1 | 1577175 | 1.000 | 254 | 0.955 |
| CLK-FPGA03 | 4059162 | 1 | 583 | 1 | 4060708 | 1.000 | 558 | 0.957 |
| CLK-FPGA04 | 2716961 | 1 | 380 | 1 | 2717722 | 1.000 | 367 | 0.966 |
| CLK-FPGA05 | 3532759 | 1 | 569 | 1 | 3533407 | 1.000 | 534 | 0.938 |
| CLK-FPGA06 | 4485498 | 1 | 591 | 1 | 4486401 | 1.000 | 572 | 0.968 |
| CLK-FPGA07 | 1708920 | 1 | 304 | 1 | 1708954 | 1.000 | 293 | 0.964 |
| CLK-FPGA08 | 1355308 | 1 | 247 | 1 | 1354247 | 0.999 | 244 | 0.988 |
| CLK-FPGA09 | 1946225 | 1 | 327 | 1 | 1945948 | 1.000 | 313 | 0.957 |
| CLK-FPGA10 | 3505733 | 1 | 512 | 1 | 3506732 | 1.000 | 499 | 0.975 |
| CLK-FPGA11 | 3270338 | 1 | 455 | 1 | 3270689 | 1.000 | 440 | 0.967 |
| CLK-FPGA12 | 2592324 | 1 | 409 | 1 | 2593721 | 1.001 | 395 | 0.966 |
| CLK-FPGA13 | 2927103 | 1 | 441 | 1 | 2926786 | 1.000 | 420 | 0.952 |
| Average | | 1.000 | | 1.000 | | 1.000 | | 0.962 |

Comparison of HPWL and running time (s) before and after applying the two-step clock constraint legalization (CCL)

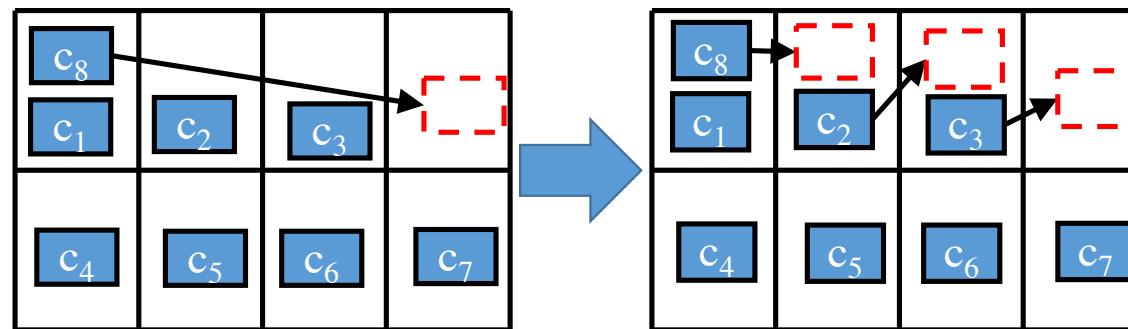
Legalization

- Refined-Tetris-Based BLE Legalization
 - Start from a small window
 - Sites inside a window are consider to have same displacement
 - Sort the sites by an objective function (HPWL)
 - If cannot be legalized, slowly increase the window size until it's legalized
 - Keep BLEs intact unless cannot be legalized



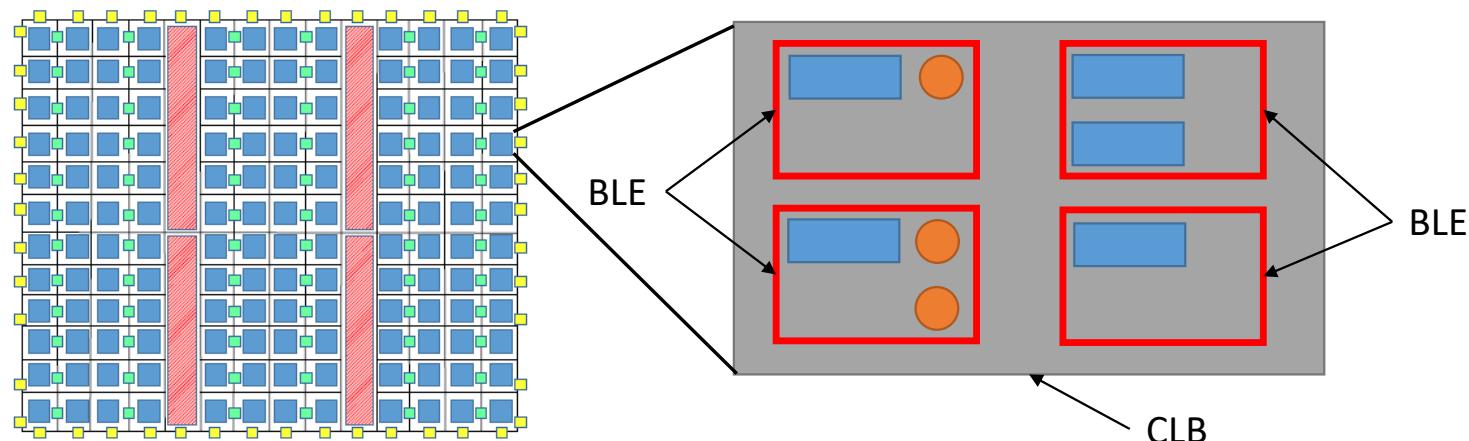
Legalization

- Chain Move
 - Invoked when the displacement of c_i is larger than D_{max}
 - The chain move should satisfy:
 - The total displacement should be no larger than D_{max}
 - The displacement of each cell should be no larger than D_{max}



Detailed Placement

- Move to **optimal region** to optimize HPWL
 - In both **BLE** level and **CLB** level
 - In **CLB** level, if the site is occupied, swap the cells if the HPWL can be decreased
 - In **BLE** level, a BLE can only be moved to a site if no rules are violated (still legalized)



Detailed Placement

- If a cell is already in its optimal region, move it to site to optimize alignment.

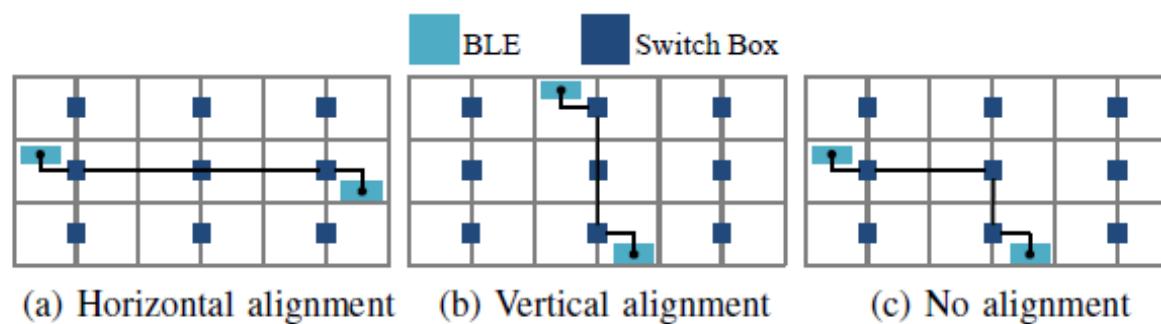
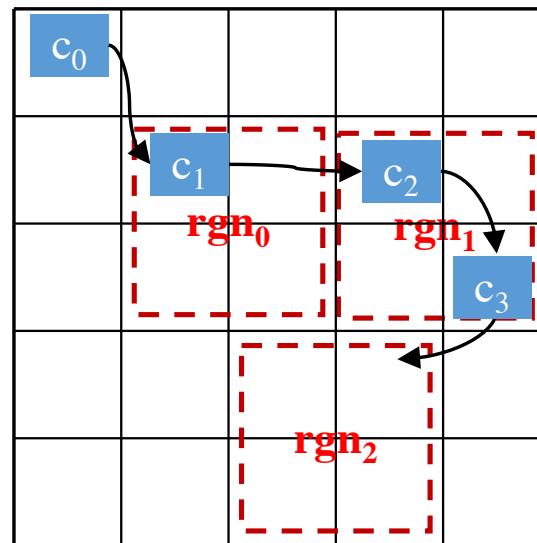


Fig. 11: Three cases with the same switch-box HPWL but different alignment.

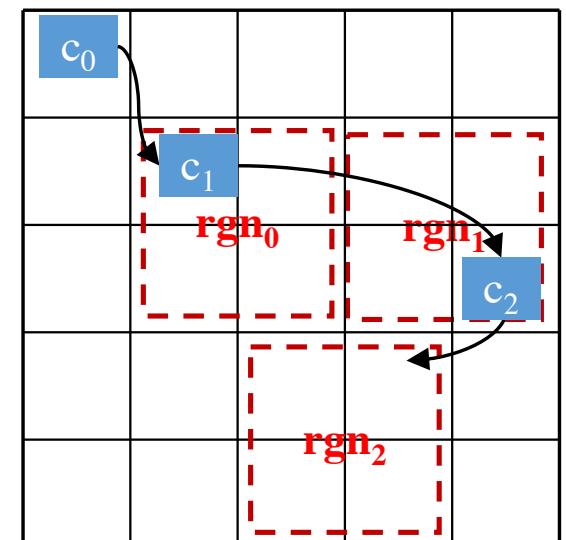
Detailed Placement – Chain Move

- Chain Move
 - Reduce the distance to optimal region in detailed placement
 - The candidate positions of each cell are those in the cell's optimal region



Detailed Placement – Chain Move

- Generate a sequence of cell moves such that
 - All the cells involved are still legal after the move
 - The objective is further optimized
- DFS-based
 - Limit the number of trials for each cell and the length of the chain
- General framework and easy to modify



Outline

- ASIC Placement and Basics
- FPGA Placement
- AI Engine Accelerated Placer
- Detailed Placement with Multi-row Cells

DREAMPlace

Objective of nonlinear placement

$$\begin{aligned} \min_{\mathbf{x}, \mathbf{y}} \quad & WL(\mathbf{x}, \mathbf{y}), \\ \text{s.t.} \quad & D(\mathbf{x}, \mathbf{y}) \leq t_d \end{aligned}$$



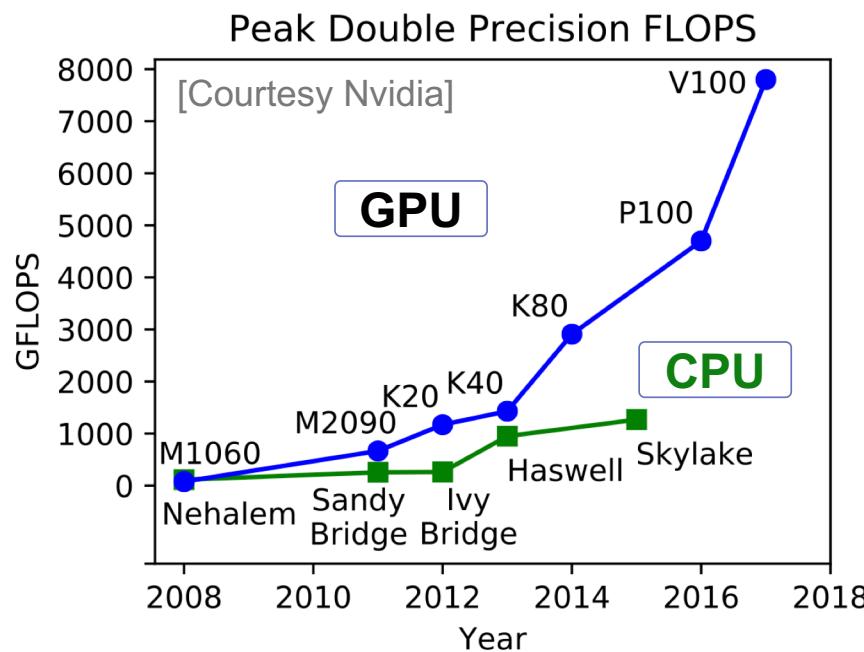
$$\min \underbrace{\left(\sum_{e \in E} WL(e; \mathbf{x}, \mathbf{y}) \right)}_{\text{Wirelength}} + \lambda \underbrace{D(\mathbf{x}, \mathbf{y})}_{\text{Density}}$$

Challenges of Nonlinear Placement

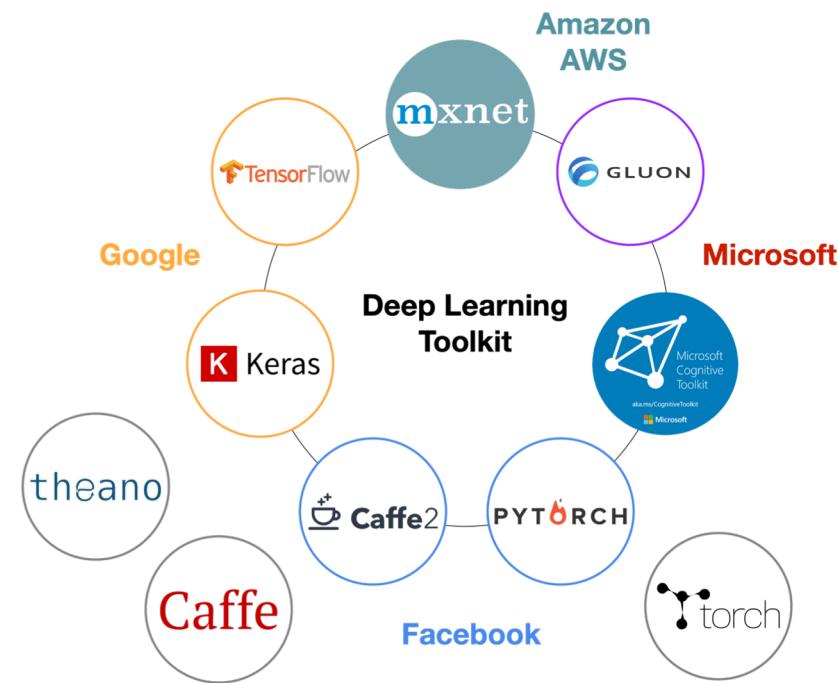
Low efficiency

Long development cycle

Advances in Deep Learning Hardware/Software



Over **60x** speedup in neural network training since 2013

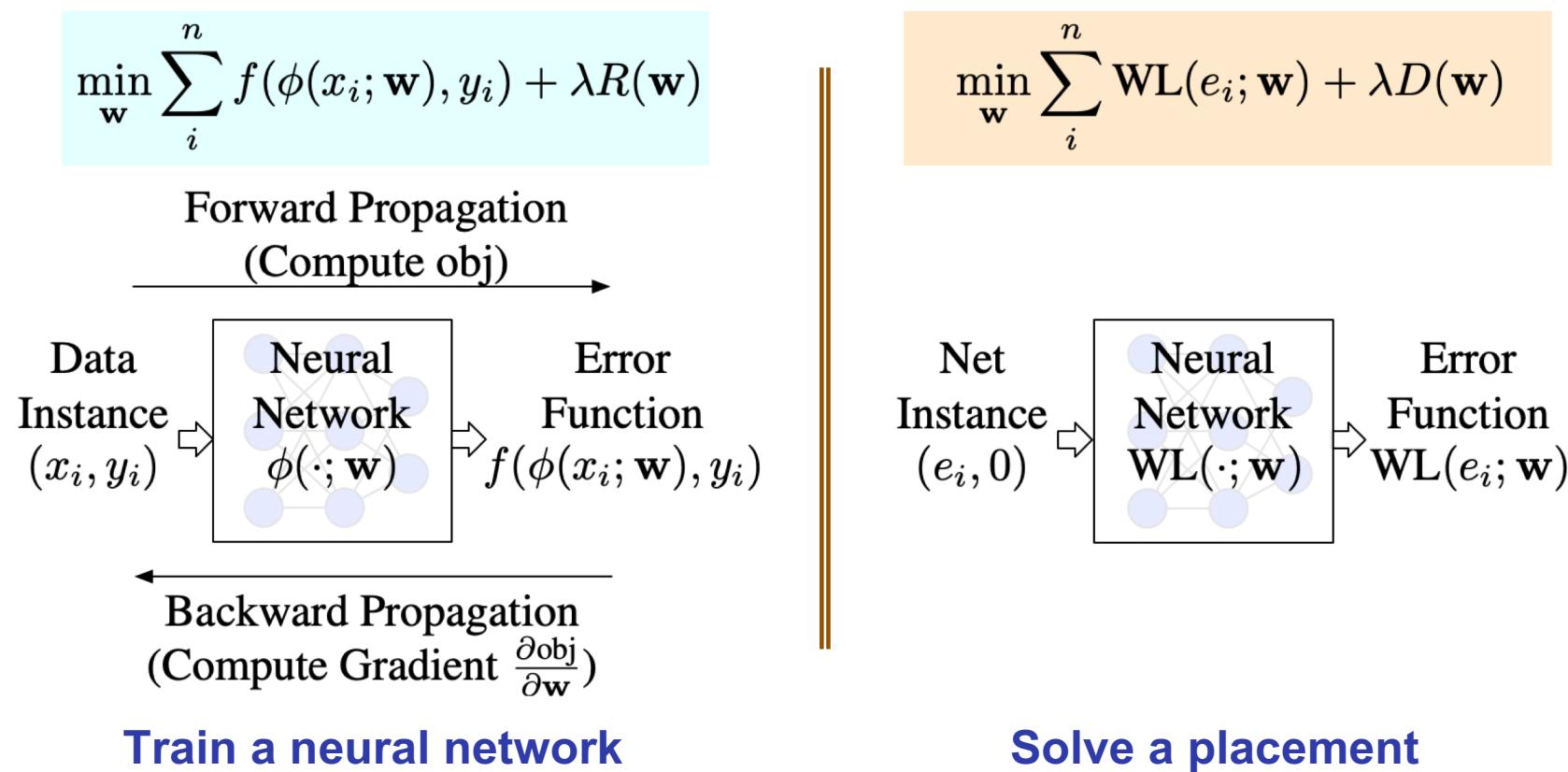


Deep learning toolkits

DREAMPlace Ideas

- We propose a novel **analogy** by casting the nonlinear placement optimization into a neural network training problem
- Greatly leverage deep learning hardware (GPU) and software toolkit (e.g., PyTorch)
- Enable ultra-high parallelism and acceleration while getting the state-of-the-art results

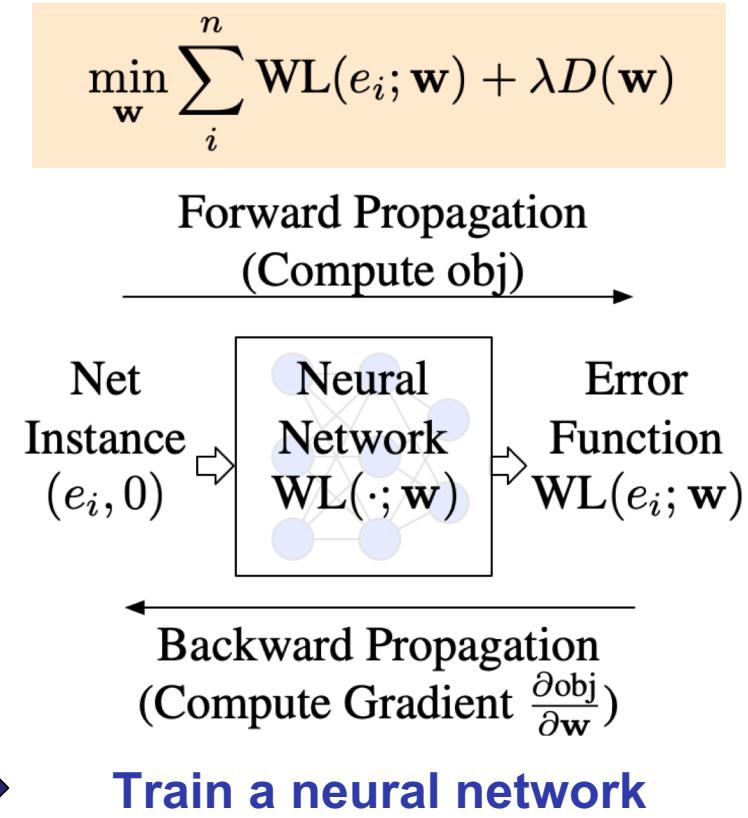
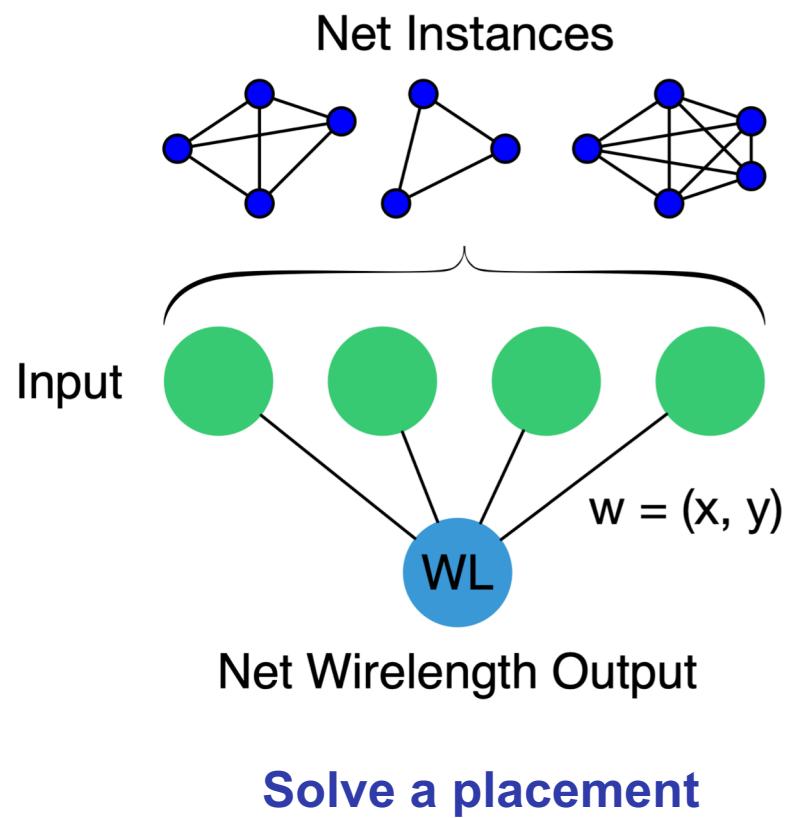
Analogy between NN-Training and Placement



優化placement的過程其實跟machine learning的過程很雷同，其實有很多machine learning的東西可以直接套用在placement上面

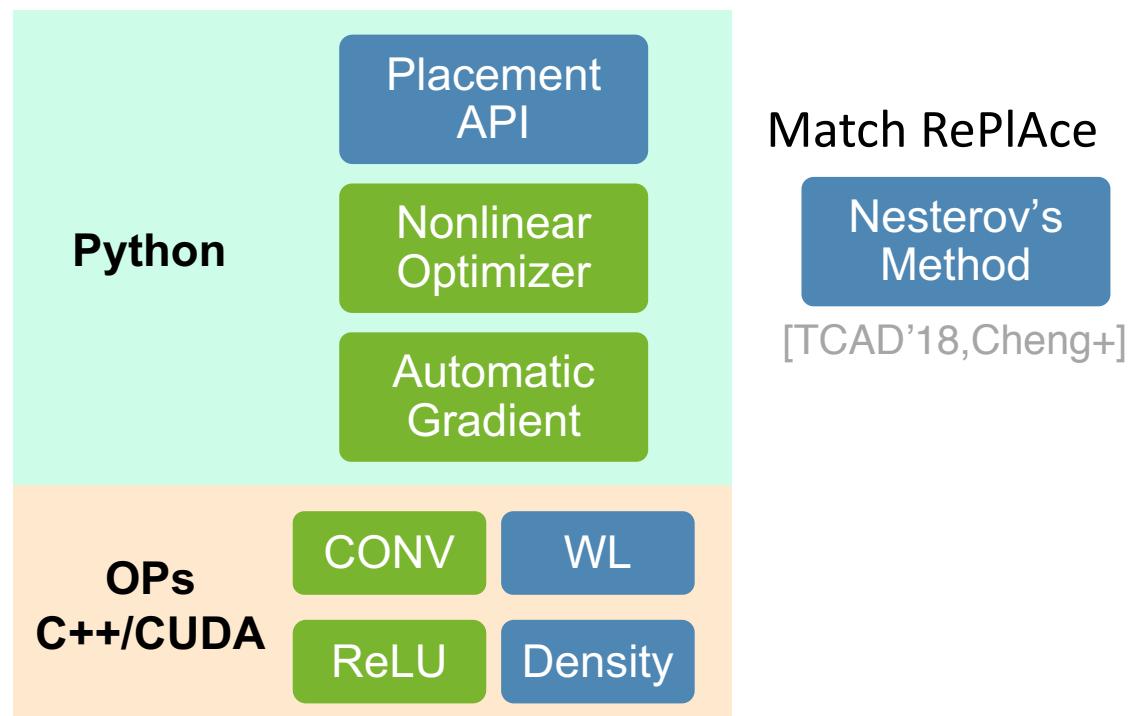
Analogy between NN Training and Placement

Casting the placement problem into neural network training



DREAMPlace Architecture

- Leverage highly optimized deep learning toolkit PyTorch



Comparisons

DREAMPlace

- CPU: Intel E5-2698 v4 @2.20GHz
- GPU: 1 NVIDIA Tesla V100
- Single CPU thread was used

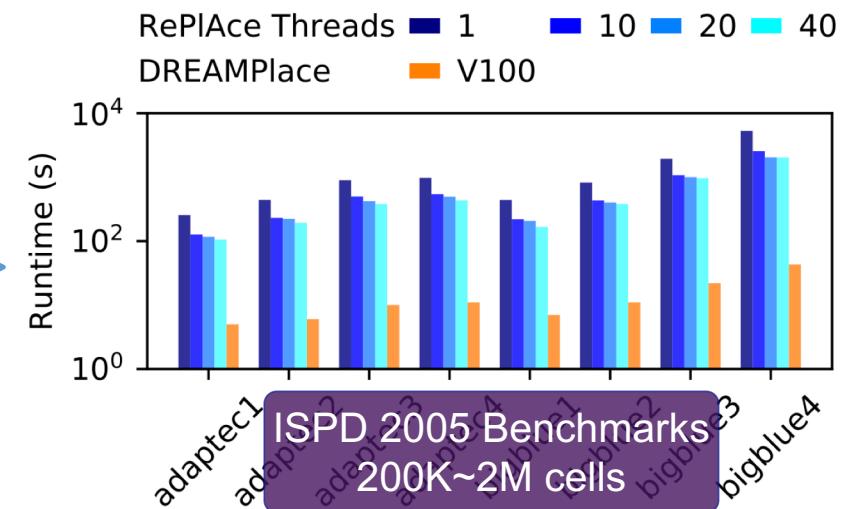
RePIAPlace [TCAD'18, Cheng+]

- CPU: 24-core 3.0 GHz Intel Xeon
- 64GB memory allocated

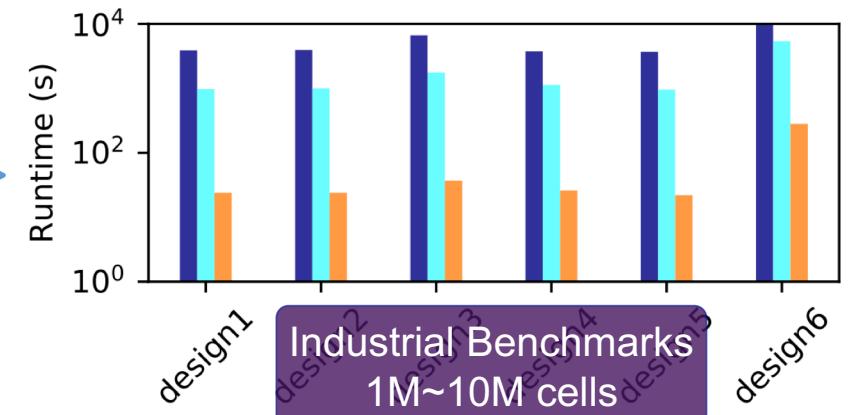
Same quality of results!

10M-cell design
finishes within **5min c.f. 3h**

34x speedup



43x speedup



Outline

- ASIC Placement and Basics
- FPGA Placement
- AI Engine Accelerated Placer
- Detailed Placement with Multi-row Cells

Detailed Placement with Multi-row Cells

Motivation

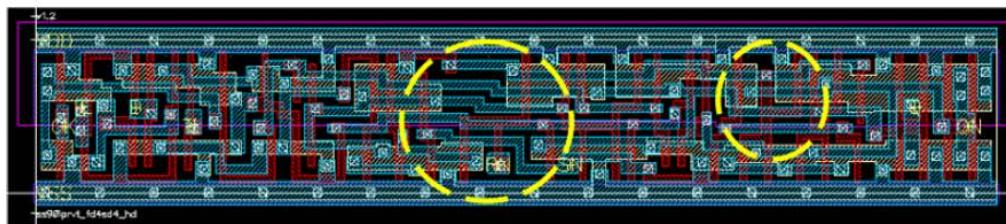


Figure 1: Large Single-Row Cell [Baek et al. 2008]

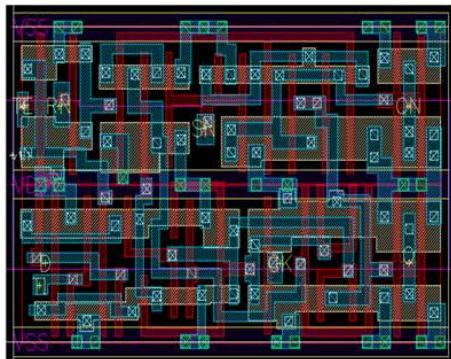


Figure 2: Multi-Row Cell [Baek et al. 2008]

- ▶ Cells like multi-bit flip-flops (MBFFs) occupy multiple rows ^a.
- ▶ Cells are much more accessible by being modified to be multi-row height ^b.

^a[Lin, Hsu, and Chang 2011]

^b[Raghavan et al. 2016]

Legalization

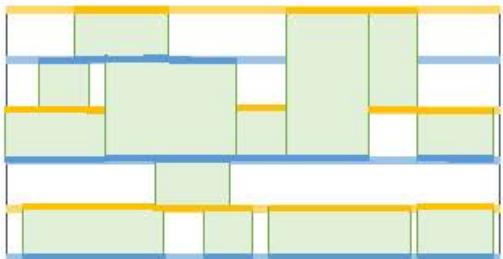


Figure 3: Power/ground alignment.

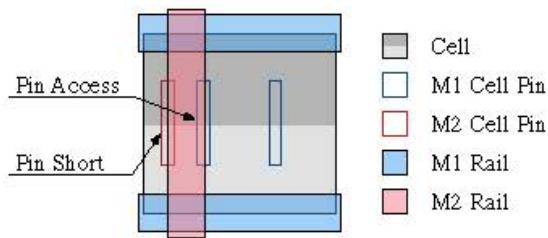


Figure 4: Pin access and pin short.

Objective function ^a:

$$S_{am} = \frac{1}{H} \sum_{h=1}^H \frac{1}{|C_h|} \sum_{c_i \in C_h} \delta_i, \quad (1)$$

where $\delta_i = \delta_{xi} + \delta_{yi} = |x_i - x'_i| + |y_i - y'_i|$, satisfying ^b:

- ▶ Cells are overlap-free;
- ▶ Cells are aligned to placement sites.
- ▶ Cells with height of even multiples of site height must be placed in alternate rows with matching power and ground alignment.
- ▶ Signal pins of cells should not be short or inaccessible due to the P/G grids and IO pins ^c.

^a[Darav, Bustany, et al. 2017]

^b[Chow, Pui, and Young 2016]

^c[Darav, Kennings, et al. 2016]

Detailed Placement Flow

The detailed placement consists of three stages.

- ▶ Inserts the cells sequentially into the placement region.
- ▶ Optimize the maximum displacement by swapping cells.
- ▶ Further optimize the average and maximum displacement.

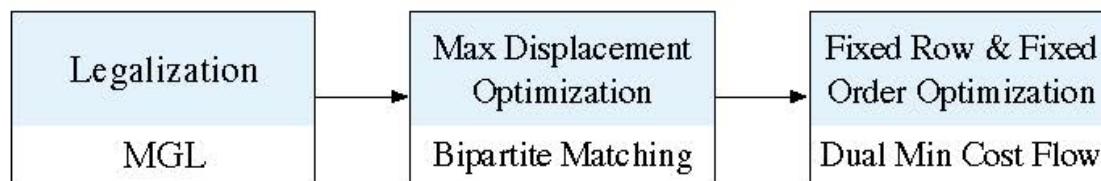
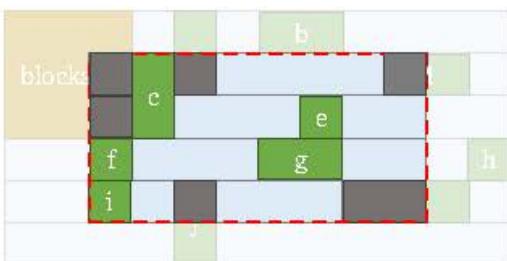


Figure 5: Detailed placement flow.

[Chow, Pui, and Young 2016]



- ▶ Define local region
- ▶ Enumerate insertion points
- ▶ Evaluate cost
- ▶ Spread overlapping cells

Figure 6: Local Region

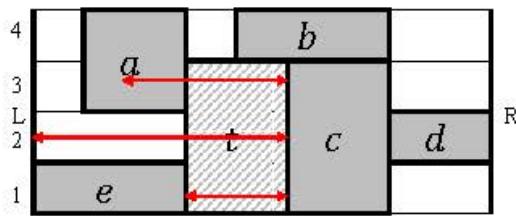


Figure 7: Insertion Point

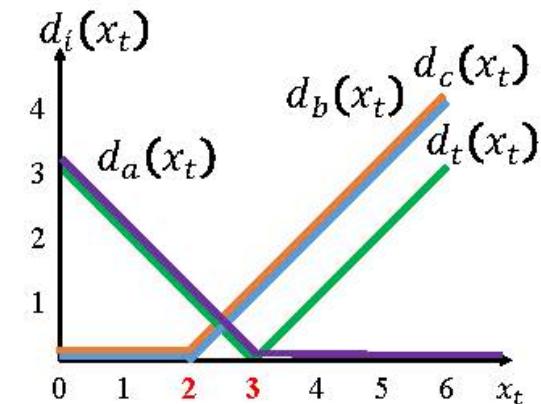


Figure 8: Cost Evaluation

Difference between MLL & MGL

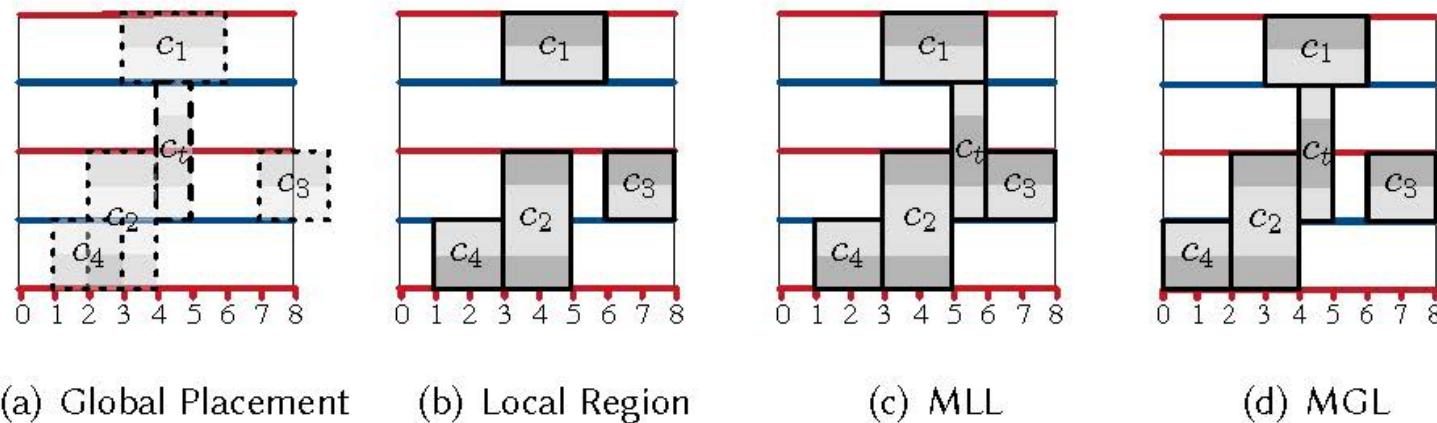
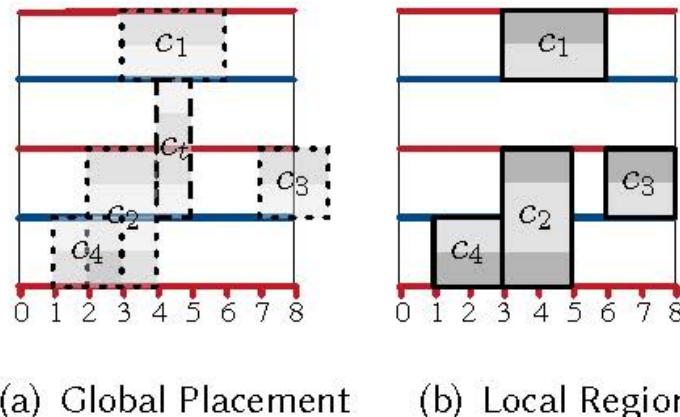


Figure 9: Comparison between MLL and MGL.

MLL optimizes the total displacement from the initial positions of the cells in the window before calling MLL.

MGL minimizes the displacement from the respective positions of cells obtained after global placement.

Clustered Cells



(a) Global Placement

(b) Local Region

Figure 10: c_2 and c_4 form a cluster.

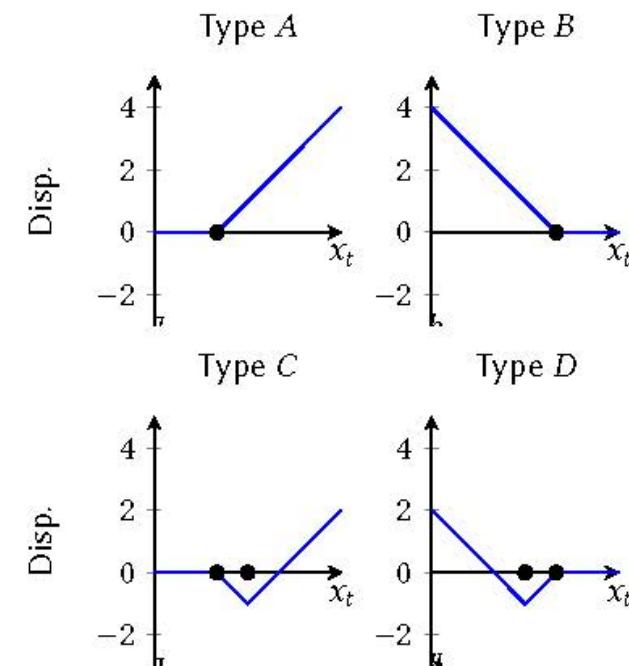


Figure 11: Cost Curve Types A – D.

Detailed Placement Flow

The detailed placement consists of three stages.

- ▶ Inserts the cells sequentially into the placement region.
- ▶ Optimize the maximum displacement by swapping cells.
- ▶ Further optimize the average and maximum displacement.

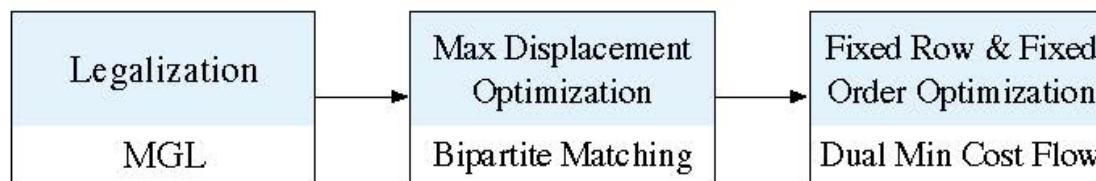


Figure 5: Detailed placement flow.

Bipartite Matching

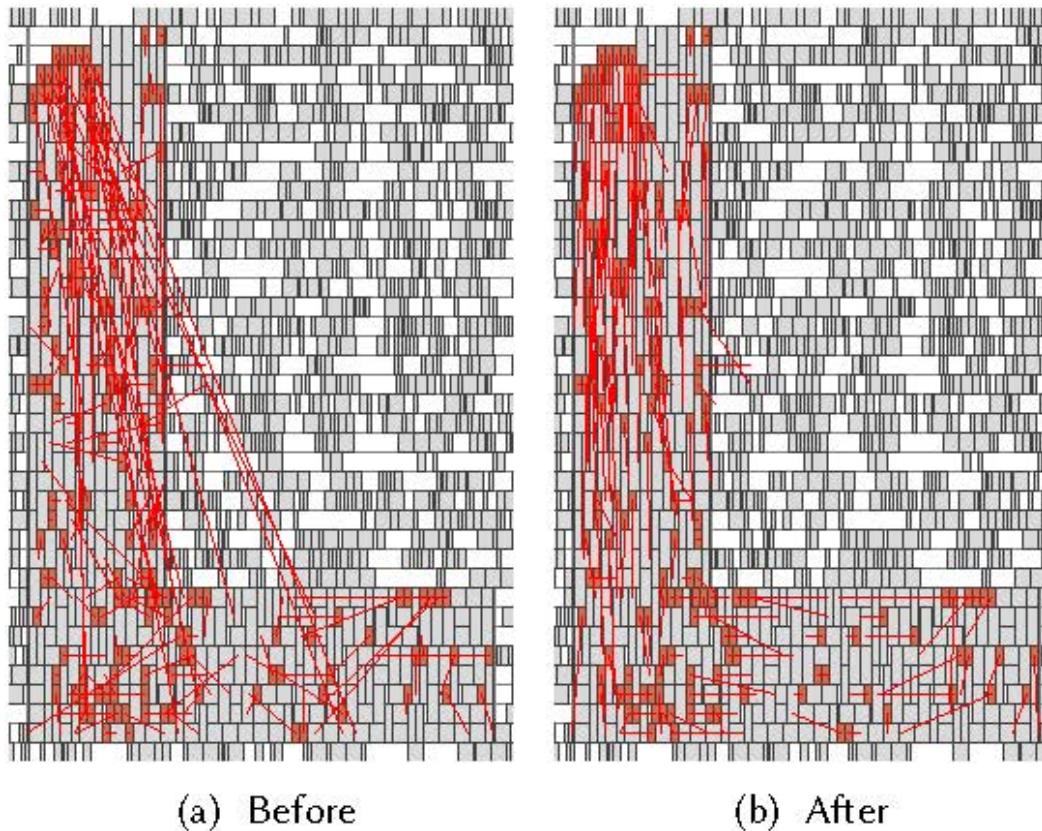


Figure 12: Bipartite matching.

Fixed Row & Fixed Order Optimization

Formulate linear displacement:

$$\max_{x_i, x_i^-, x_i^+} \sum_i n_i(x_i^- - x_i^+) \quad (2)$$

$$\text{s.t. } x_i^- \leq x_i - x'_i \leq x_i^+, \quad \forall c_i \in C, \quad (2a)$$

$$x_i^- \leq 0 \leq x_i^+, \quad \forall c_i \in C, \quad (2b)$$

$$x_i - x_j \leq -w_i, \quad \forall (i, j) \in E, \quad (2c)$$

$$l_i \leq x_i \leq r_i, \quad \forall c_i \in C. \quad (2d)$$

Let \tilde{x}^0 be the absolute position of the *origin*, then the absolute positions $\{\tilde{x}_i, \tilde{x}_i^-, \tilde{x}_i^+\}$ of $\{x_i, x_i^-, x_i^+\}$ are

$$\tilde{x}_i = x_i + \tilde{x}^0, \tilde{x}_i^- = x_i^- + \tilde{x}^0, \tilde{x}_i^+ = x_i^+ + \tilde{x}^0. \text{ Thus,}$$

$$\max_{\tilde{x}_i, \tilde{x}_i^-, \tilde{x}_i^+, \tilde{x}^0} \sum_i n_i(\tilde{x}_i^- - \tilde{x}_i^+) \quad (3)$$

$$\text{s.t. } \tilde{x}_i^- \leq \tilde{x}_i - x'_i \leq \tilde{x}_i^+, \quad \forall c_i \in C, \quad (3a)$$

$$\tilde{x}_i^- - \tilde{x}^0 \leq 0 \leq \tilde{x}_i^+ - \tilde{x}^0, \quad \forall c_i \in C, \quad (3b)$$

$$\tilde{x}_i - \tilde{x}_j \leq -w_i, \quad \forall (i, j) \in E, \quad (3c)$$

$$l_i \leq \tilde{x}_i - \tilde{x}^0 \leq r_i, \quad \forall c_i \in C, \quad (3d)$$

whose dual linear programming is a min-cost flow problem.

Example of Min-Cost Flow

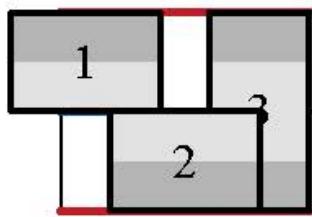


Figure 13: GP.

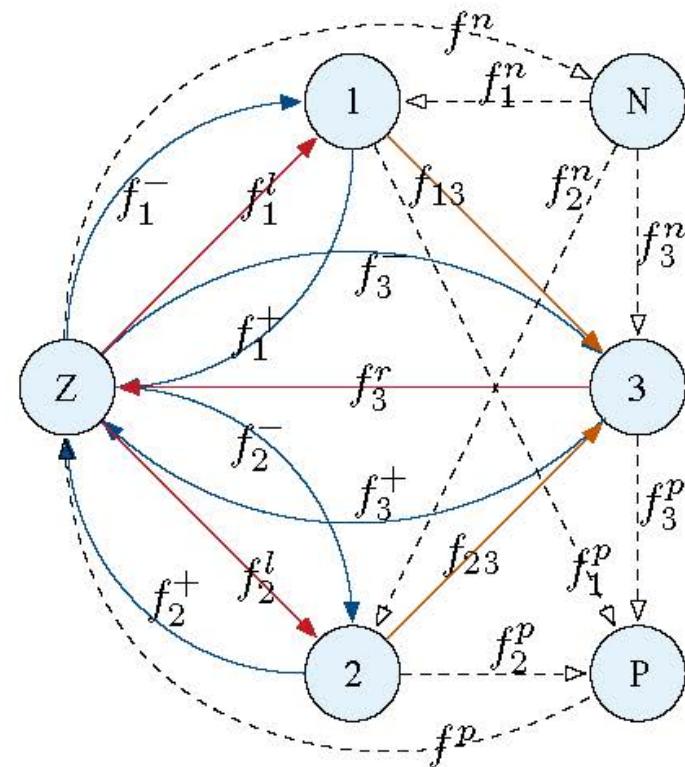


Figure 14: Flow network.

Outline

- ASIC Placement and Basics
- FPGA Placement
- AI Engine Accelerated Placer
- Detailed Placement with Multi-row Cells
- Q&A

Thank You