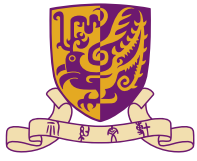
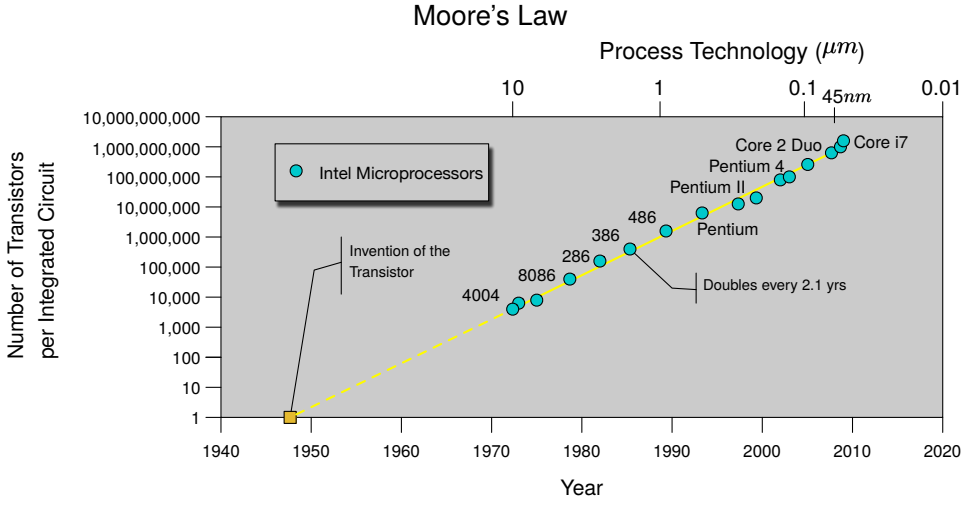
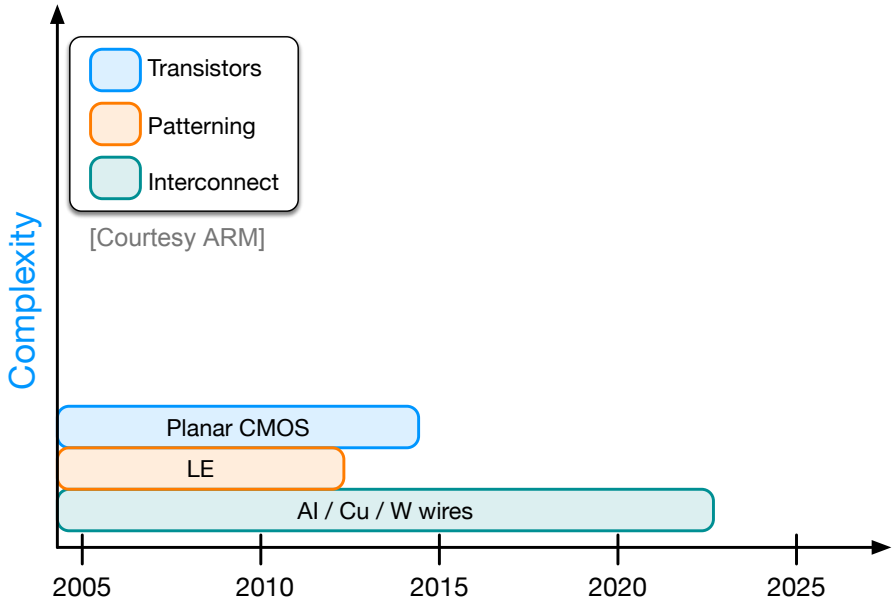
OpenMPL:
An Open Source Layout Decomposer

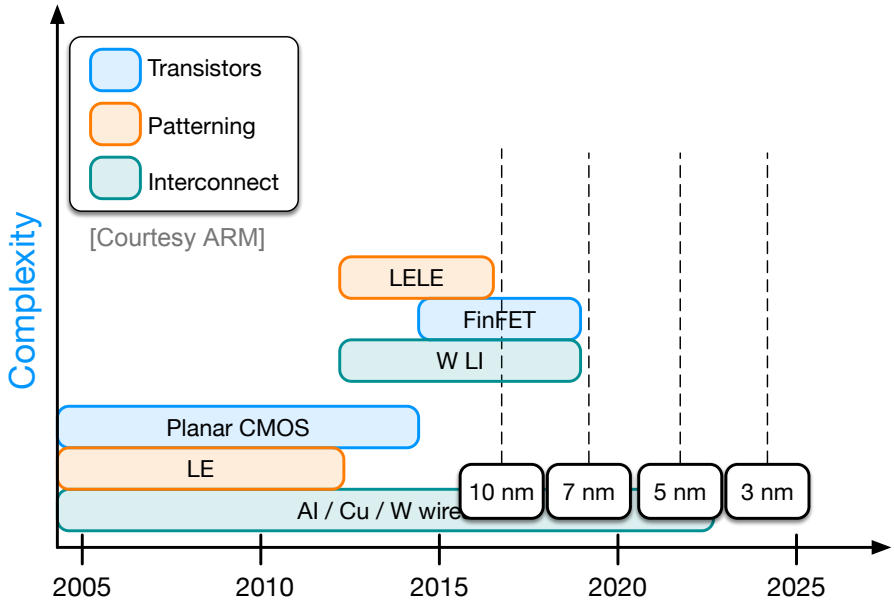
Wei Li[1], Yuzhe Ma[1], Qi Sun[1], **Yibo Lin**[2], Iris Hui-Ru Jiang[3],
Bei Yu[1], David Z. Pan[4]

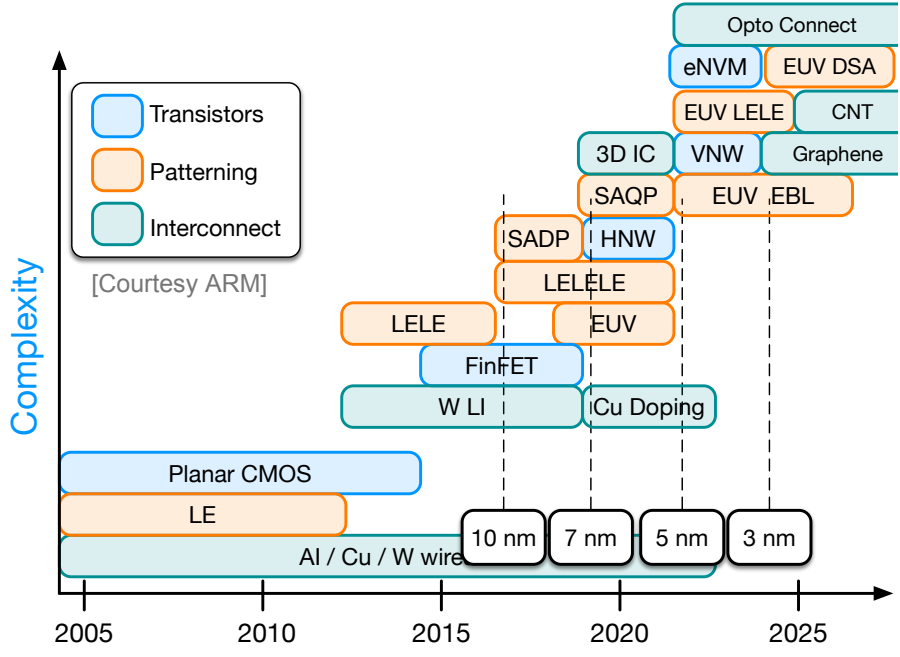
[1]The Chinese University of Hong Kong, [2]Peking University,
[3]National Taiwan University, [4]University of Texas at Austin

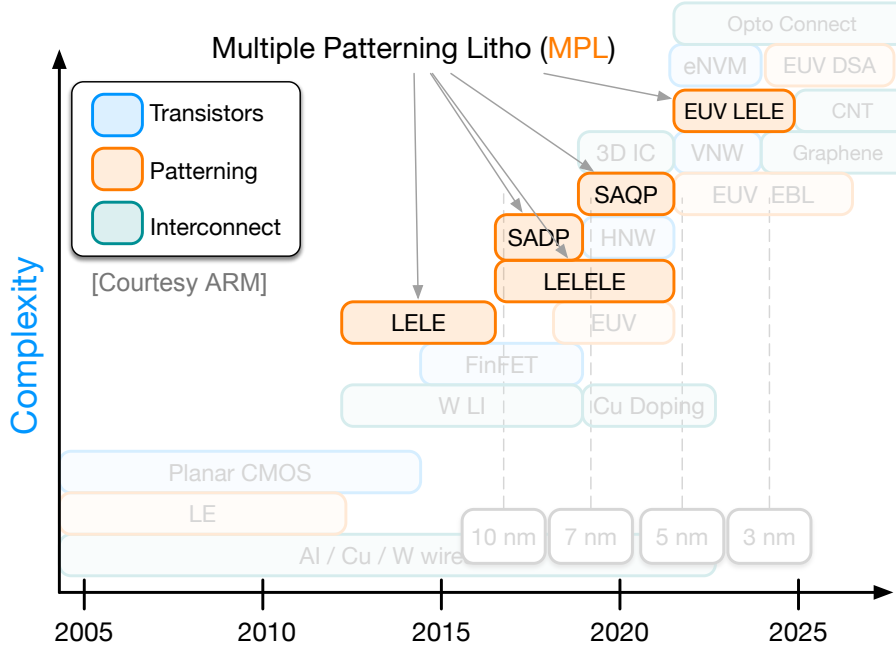



Moore's Law
Process Technology (μm)
10
1
0.1
$45nm$
0.01
Number of Transistors per Integrated Circuit
10,000,000,000
1,000,000,000
100,000,000
10,000,000
1,000,000
100,000
10,000
1,000
100
10
1
Intel Microprocessors
Invention of the Transistor
4004
8086
286
386
486
Pentium
Pentium II
Pentium 4
Core 2 Duo
Core i7
Doubles every 2.1 yrs
1940
1950
1960
1970
1980
1990
2000
2010
2020
Year

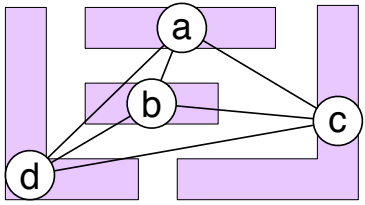

Transistors
Patterning
Interconnect
[Courtesy ARM]
Complexity
Planar CMOS
LE
Al / Cu / W wires
2005
2010
2015
2020
2025


Transistors
Patterning
Interconnect
[Courtesy ARM]
Complexity
LELE
FinFET
W LI
Planar CMOS
LE
Al / Cu / W wire
10 nm
7 nm
5 nm
3 nm
2005
2010
2015
2020
2025


Opto Connect
eNVM
EUV DSA
EUV LELE
CNT
3D IC
VNW
Graphene
SAQP
EUV EBL
SADP
HNW
LELELE
LELE
EUV
FinFET
W LI
Cu Doping
Planar CMOS
LE
Al / Cu / W wire
10 nm
7 nm
5 nm
3 nm
Transistors
Patterning
Interconnect
[Courtesy ARM]
Complexity
2005
2010
2015
2020
2025

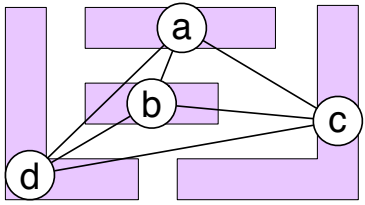

Multiple Patterning Litho (MPL)
Transistors
Patterning
Interconnect
[Courtesy ARM]
Complexity
2005
2010
2015
2020
2025
Opto Connect
eNVM
EUV DSA
EUV LELE
CNT
3D IC
VNW
Graphene
SAQP
EUV EBL
SADP
HNW
LELELE
LELE
EUV
FinFET
W LI
Cu Doping
Planar CMOS
LE
Al / Cu / W wire
10 nm
7 nm
5 nm
3 nm

Multiple Patterning Layout Decomposition

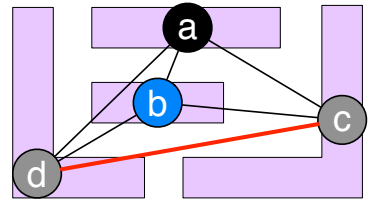


(a) Original layout

Multiple Patterning Layout Decomposition

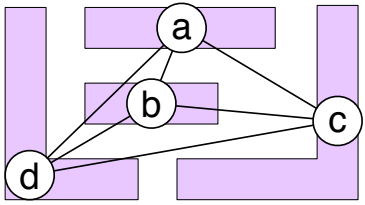


(a) Original layout

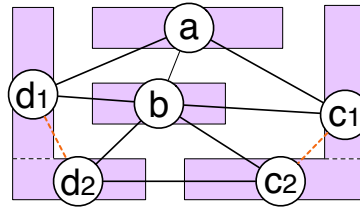


(b) TPL layout with conflicts

Multiple Patterning Layout Decomposition

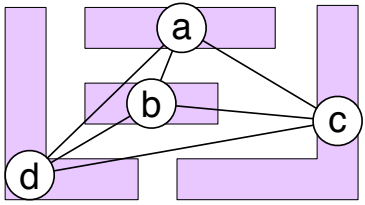


(a) Original layout

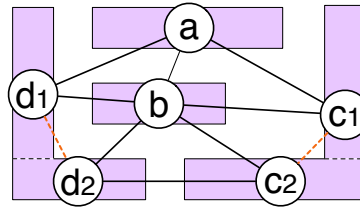


(b) Layout graph

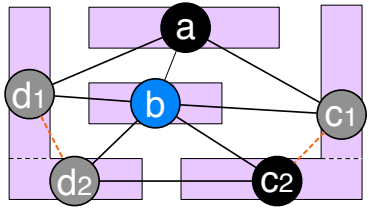
Multiple Patterning Layout Decomposition



(a) Original layout

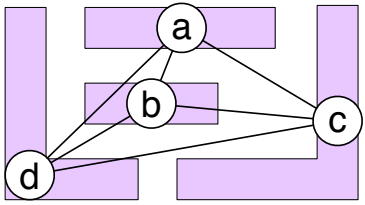


(b) Layout graph

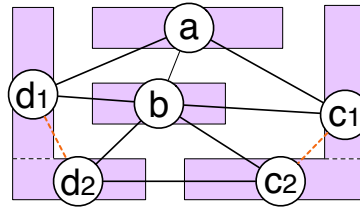


(c) Coloring on layout graph

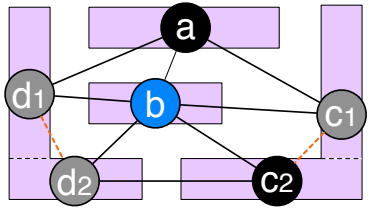
Multiple Patterning Layout Decomposition



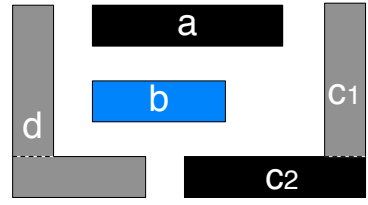
(a) Original layout



(b) Layout graph



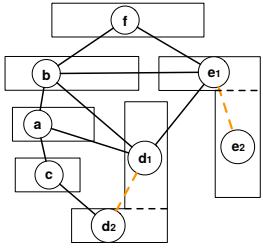
(c) Coloring on layout graph



(d) Final decomposed layout

Mathematical Formulation

- Some conflicts can be solved by Stitch Insertion.



k Pattern Layout Decomposition

$$\min_{\boldsymbol{x}} \sum_{e_{ij} \in CE} c_{ij} + \alpha \times \sum_{e_{ij} \in SE} s_{ij},$$

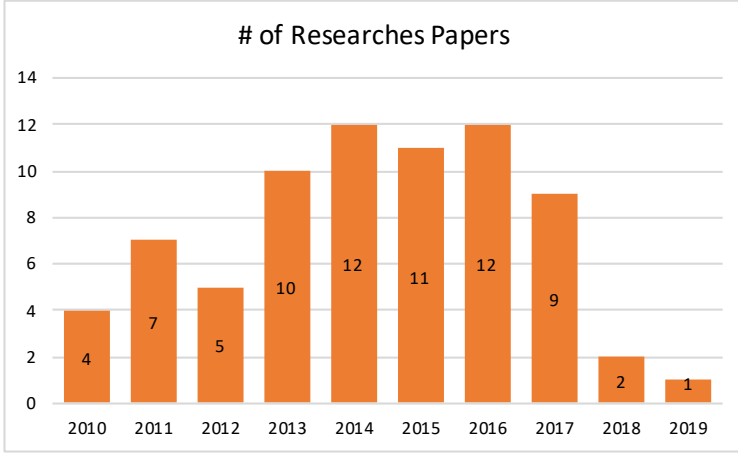
$$\begin{aligned} \text{s.t. } & c_{ij} = \{x_i == x_j\}, && \forall e_{ij} \in CE, \\ & s_{ij} = \{x_i \neq x_j\}, && \forall e_{ij} \in SE, \\ & x_i \in \{0, 1, ..., k\}, && \forall i \in V. \end{aligned}$$

- x_i is a variable for the k available colors of the pattern v_i.
- c_{ij} is a binary variable representing conflict edge $e_{ij} \in CE$.
- s_{ij} stands for stitch edge $e_{ij} \in SE$.

Research Trend for Layout Decomposition

An **UNDERESTIMATION** from Google Scholar

Google Scholar allintitle: "layout decomposition"



Challenges in Layout Decomposition Research

Complicated workflow and huge development overhead

- GDSII parsing and writing
- Layout graph construction and stitch insertion
- Graph simplification and core solvers

Challenges in Layout Decomposition Research

Complicated workflow and huge development overhead

- GDSII parsing and writing
- Layout graph construction and stitch insertion
- Graph simplification and core solvers

Lack of reusable code and repeatability

- Need to develop from scratch
- Only share executable binaries instead of source code

Challenges in Layout Decomposition Research

Complicated workflow and huge development overhead

- GDSII parsing and writing
- Layout graph construction and stitch insertion
- Graph simplification and core solvers

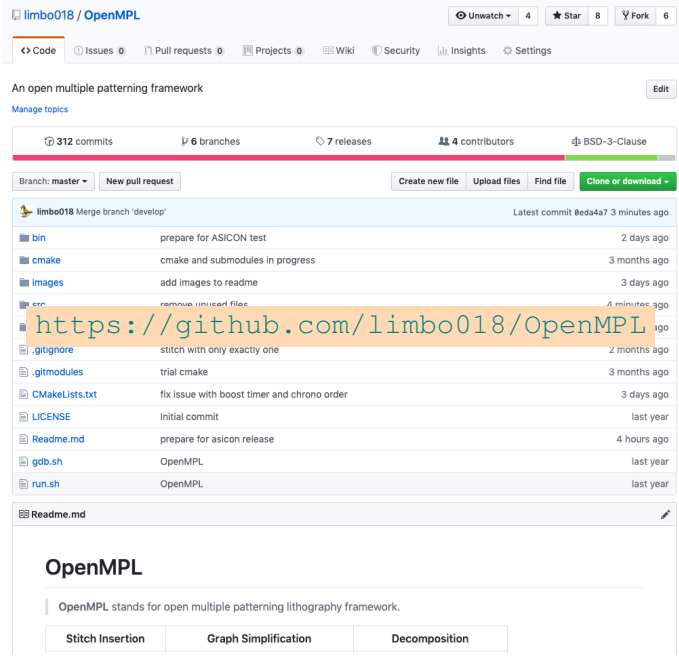
Lack of reusable code and repeatability

- Need to develop from scratch
- Only share executable binaries instead of source code

No consistent framework for fair comparison

- Different implementations to the same algorithm
- Hard to make detailed comparison in depth

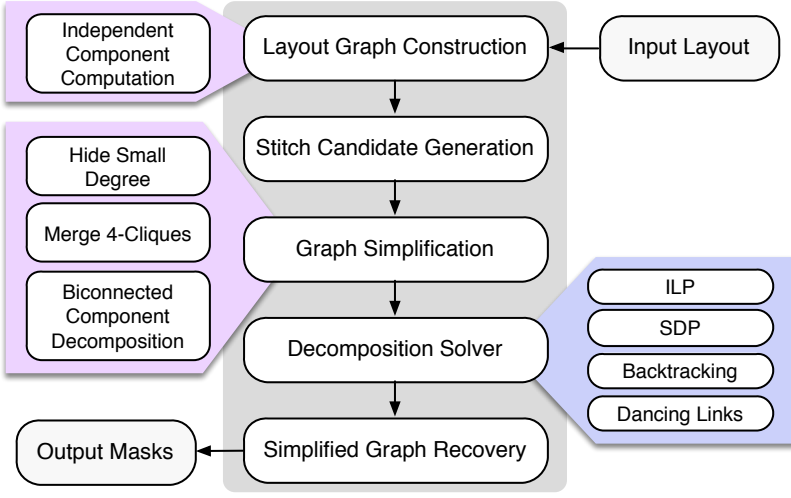
OpenMPL 2.0: Open-Source Layout Decomposition Tool



https://github.com/limbo018/OpenMPL

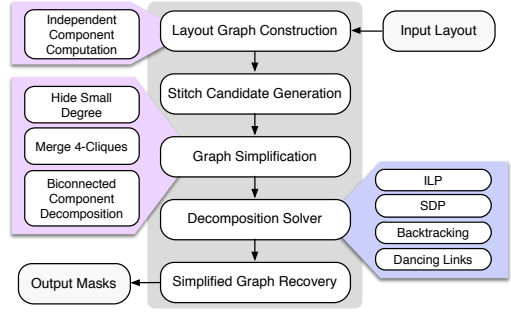
SCAN ME

Workflow


Independent Component Computation
Layout Graph Construction
Input Layout
Stitch Candidate Generation
Hide Small Degree
Merge 4-Cliques
Biconnected Component Decomposition
Graph Simplification
ILP
SDP
Backtracking
Dancing Links
Decomposition Solver
Output Masks
Simplified Graph Recovery

Design Principles

- **Decoupled design stages**
- **Graphs for communications among kernel stages**
- **Efficiency and generality for different mask data**



Extensible API to Incorporate New Algorithms

- Create a solver

```
// A solver class
class InstantColoring
    : public Coloring
{
  public:
    // g is a boost::graph
    InstantColoring(g);
  protected:
    virtual double coloring()
    {
      // override base function
      // return cost
    }
};
```

Extensible API to Incorporate New Algorithms

- Create a solver

```
// A solver class
class InstantColoring
    : public Coloring
{
  public:
    // g is a boost::graph
    InstantColoring(g);
  protected:
    virtual double coloring()
    {
      // override base function
      // return cost
    }
};
```

- Integrate the solver

```
// define a solver
switch (algorithm)
{
  case BruteForce:
    solver = new BruteForce (g);
  case InstantColoring:
    solver = new InstantColoring (g);
  ...
}
...
// solve coloring
(*solver)();
```

OpenMPL 2.0: Open-Source Layout Decomposition Tool

We implement several layout decomposition solvers.

- **Backtracking**: Use DFS algorithm to find the solutions [Yu+,ICCAD'13].

OpenMPL 2.0: Open-Source Layout Decomposition Tool

We implement several layout decomposition solvers.

- **Backtracking**: Use DFS algorithm to find the solutions [Yu+,ICCAD'13].
- **Integer Linear Programming**: Convert the problem into linear programming by binary encoding of vertex colors [Yu+,ICCAD'11].
- **Dancing Links**: Treat the problem as an exact cover problem and solve it in the BFS style [Chang+,DAC'16].

OpenMPL 2.0: Open-Source Layout Decomposition Tool

We implement several layout decomposition solvers.

- **Backtracking**: Use DFS algorithm to find the solutions [Yu+,ICCAD'13].
- **Integer Linear Programming**: Convert the problem into linear programming by binary encoding of vertex colors [Yu+,ICCAD'11].
- **Dancing Links**: Treat the problem as an exact cover problem and solve it in the BFS style [Chang+,DAC'16].
- **Semidefinite Programming**: Use SDP to relax the problem and solve it in polynomial time [Yu+,ICCAD'11].

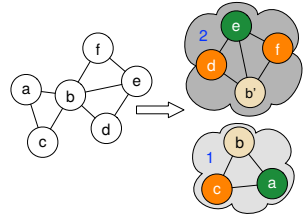
Layout Simplification

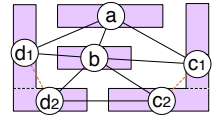
Reduce the graph size and therefore reduce the computation complexity.

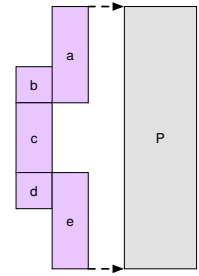
- Level 0: **No Simplification**. Do not conduct any simplification.
- Level 1: **Hide small degree**. Temporarily remove nodes with low degree [Yu+,TCAD'15] [Lin+,SPIE'16].
- Level 2: **Merge 4-Clique**. Detect and merge sub-4-clique structures [Lin+,SPIE'16].
- Level 3: **Biconnected component decomposition**. Partition original graph into independent components [Kahng+,ICCAD'08].

Additional Features

- **Multithreading**: Solve components in parallel.
- **Stitch Insertion**: Insert stitches to solve some conflicts.
- **Shape Friendly**: Specify the shape (POLYGON or RECTANGLE) easily.







Command Line Options

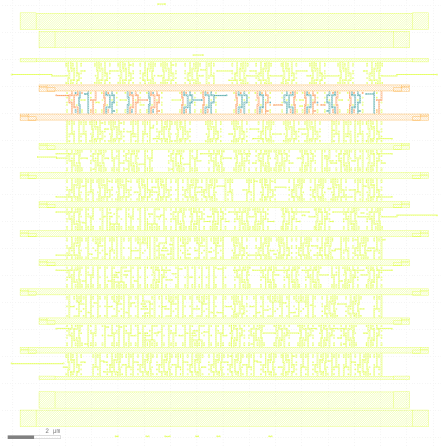
Selected Options	Descriptions
-coloring_distance	a floating point number indicating number of coloring distance in nanometer
-color_num	an integer indicating number of masks (colors) < 3|4 >
-simplify_level	an integer indicating graph simplification level < 0|1|2|3 >
-path_layer	an integer indicating layer for conflict edges
-precolor_layer	an integer indicating layer for pre-colored patterns
-uncolor_layer	an integer indicating layer for coloring
-algo	algorithm type < ILP|BACKTRACK|LP|SDP >
-shape	shape mode < RECTANGLE|POLYGON >
-use_stitch	use stitch to avoid conflict
-gen_stitch	generate stitch candidate
-weight_stitch	weight of stitch
-thread_num	an integer indicating maximum thread number
-verbose	toggle controlling screen messages
-dbg_comp_id	debug component id

Experimental Settings

- Triple Patterning Lithography
- Conducted on modified ISCAS benchmarks from [Yu+,ICCAD'13]
- Coloring distance: 120nm for the first ten cases and 100nm for the last five cases.

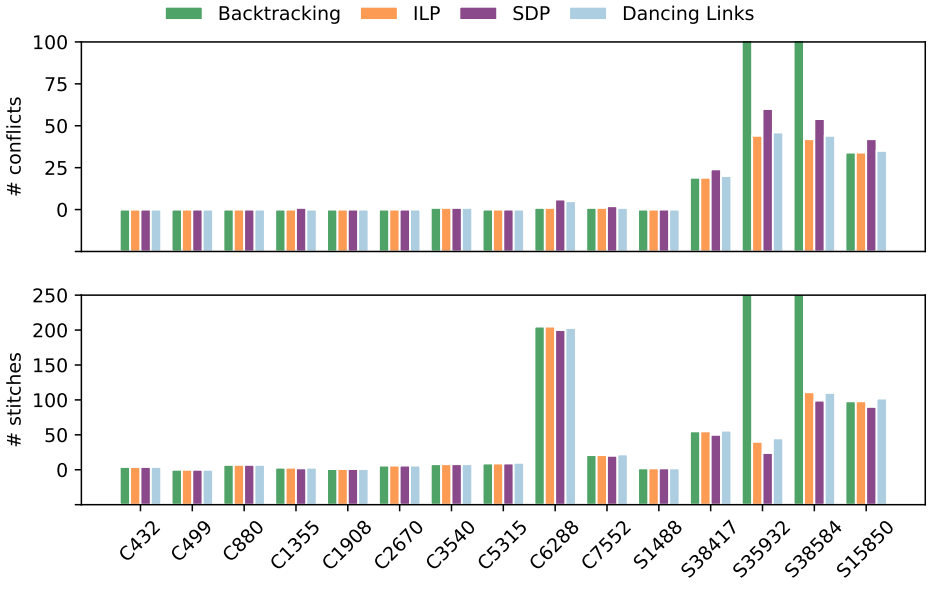
Experimental Results

(a) total_c1, TPLD, SDP coloring process.

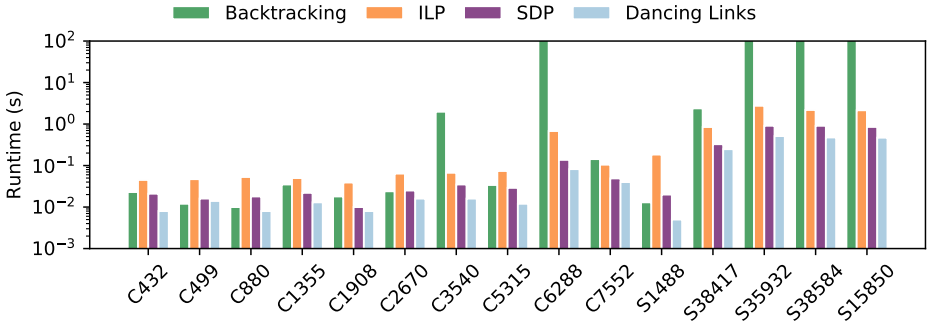


(b) total_c2, TPLD, SDP coloring process.

Experimental Results – Quality



Experimental Results – Runtime



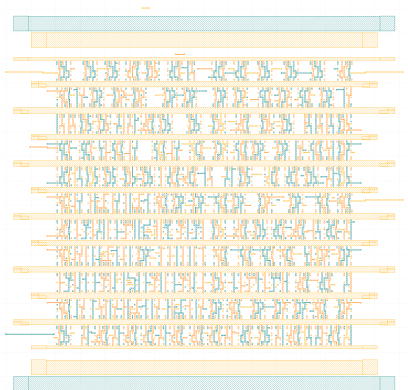
Quality and Runtime Tradeoffs

	Quality	Efficiency
Backtracking	★ ★ ★ ★ ★	★
ILP	★ ★ ★ ★ ★	★ ★
SDP	★ ★ ★	★ ★ ★ ★
Dancing Links	★ ★ ★ ★	★ ★ ★ ★ ★

Conclusion

Open-source layout decomposition framework

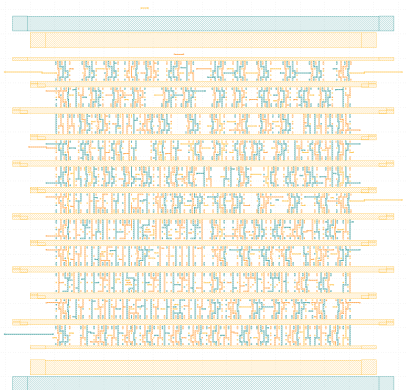
- Four layout decomposition solvers;
- Four graph simplification strategies;
- Multithreads, stitch insertion supported and shape frinedly.



Conclusion

Open-source layout decomposition framework

- Four layout decomposition solvers;
- Four graph simplification strategies;
- Multithreads, stitch insertion supported and shape frinedly.



Future Work

- Efficiency: hardware acceleration.
- Quality: post refinement.

Thank You

SCAN ME

Wei Li (wli@cse.cuhk.edu.hk)

Yuzhe Ma (yzma@cse.cuhk.edu.hk)

Qi Sun (qsun@cse.cuhk.edu.hk)

Yibo Lin (yibolin@pku.edu.cn)

Iris Hui-Ru Jiang (huiru.jiang@gmail.com)

Bei Yu (byu@cse.cuhk.edu.hk)

David Z. Pan (dpan@ece.utexas.edu)