OpenMPL: An Open Source Layout Decomposer

Qi Sun
Chinese University of Hong kong
qsun@cse.cuhk.edu.hk

Yibo Lin
University of Texas at Austin
yibolin@utexas.edu

Iris Hui-Ru Jiang
National Taiwan University
huiru.jiang@gmail.com

Bei Yu
Chinese University of Hong kong
byu@cse.cuhk.edu.hk

David Z. Pan
University of Texas at Austin
dpan@ece.utexas.edu

ABSTRACT

Multiple patterning lithography (MPL) has been widely adopted in advanced technology nodes of VLSI manufacturing. As a key step in the design flow, layout decomposition for MPL is critical to design closure. Various algorithms have been proposed to speedup layout decomposition and meanwhile maintain the performance. However, due to the complicated design flow, heavy engineering effort for integration and tuning is required to reproduce them, raising the bar for further advancing the field. This paper presents OpenMPL [1], an open-source multiple patterning layout decomposition framework, with efficient implementations of various state-of-the-art algorithms. Experiments are conducted on widely-recognized benchmarks and promising results are demonstrated.

KEYWORDS

Multiple Patterning, Layout Decomposition, Open Source

1 INTRODUCTION

According to Moore's law, the density of transistors in integrated circuits (ICs) would double every two years, and features sizes go to extremely small due to aggressive scaling. Nevertheless, the next-generation of lithography technology is still under development which limits the production of denser integrated circuits.

Multiple patterning layout decomposition (MPLD) has been adopted for improving the lithography resolution. The key idea of MPLD is to divide features that are close to each other into different masks, such that within each mask, the features are far away enough for existing lithography technique. MPLD includes double patterning layout decomposition (DPLD), triple patterning layout decomposition (TPLD) and quadratic patterning layout decomposition (QPLD), according to the number of masks. This problem is very difficult since it is a variation to graph coloring problem, which is NP-hard for $k \geq 3$. Figure 1 is an example of TPLD.

Many algorithms have been proposed to solve MPLD, and the high speed of operation is the first target. In this paper, we present a general framework, OpenMPL, for MPLD and provide efficient implementation for state-of-the-art layout decomposition algorithms, including integer linear programming (ILP) [2–4], linear programming (LP) [5], semidefinite programming (SDP) [2, 6–9], backtracking algorithm and dancing links [10], and some heuristic methodologies [11, 12]. To help solve this problem, some speed-up techniques are incorporated, including Hiding Low Degree Nodes [2, 5, 13], Merging 4-Clique [5], and Biconnected Components Decomposition [5, 12, 14, 15]. These three speed-up techniques have been widely used and proven to be efficient.

The rest of this paper is organized as follows. Section 2 introduces several MPLD techniques, the speed-up techniques and some other methods implemented in our framework OpenMPL [1]. Section 3 lists some preliminary experimental results, followed by conclusion and future work in Section 4.

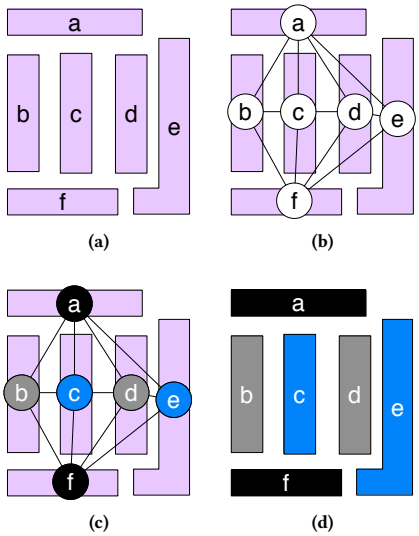

Figure 1: An example of TPLD. (a) The original layout; (b) The constructed layout graph; (c) Coloring on the layout graph; (d) The final decomposed layout with three masks (each color corresponds to one mask).

2 THE OpenMPL FRAMEWORK

OpenMPL is a general framework for various decomposition algorithms and different levels of speed-up techniques. These methods are well embedded into the framework with unified interfaces. The

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.
WOSET'18, November 8, 2018, San Diego, CA, USA
© 2018 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

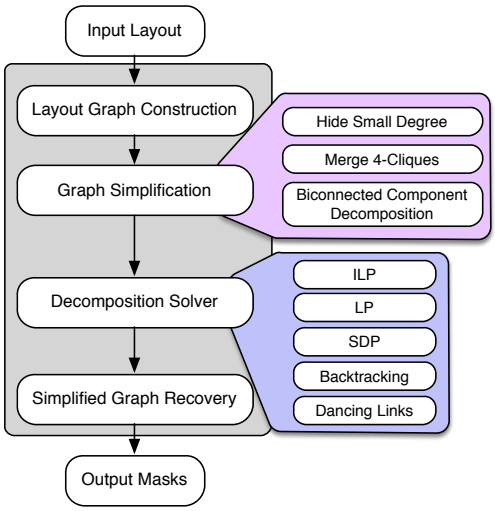

Figure 2: The workflow of OpenMPL.

framework operation flow is in Figure 2. All the algorithms are implemented in C++ and can be deployed on Linux. Technical details are as follows.

2.1 Layout Decomposition Methods

For a given undirected layout graph $G = (V, E)$, every node $v_i \in V$ corresponds one pattern in circuit, each edge $e_{ij} \in E$ is used to characterize the relationships between patterns (conflict or stitch). Further, $E = \{CE \cup SE\}$, where SE is the set of stitch edges and CE is the set of conflict edges. The MPLD problem can be formulated as below:

$$\min_{x} \sum c_{ij} + \sum s_{ij}, \tag{1a}$$

$$\text{s.t. } c_{ij} = (x_i == x_j), \qquad \forall e_{ij} \in CE, \tag{1b}$$

$$s_{ij} = x_i \oplus x_j, \qquad \forall e_{ij} \in SE, \tag{1c}$$

$$x_i \in \{0, 1, \ldots, k\}, \qquad \forall i \in V, \tag{1d}$$

where x_i is a variable for the k available colors of the pattern v_i, c_{ij} is a binary variable representing conflict edge $e_{ij} \in CE$ and s_{ij} stands for stitch edge $e_{ij} \in SE$. If two nodes, x_i and x_j, within the minimal coloring distance are assigned the same color (i.e. $x_i == x_j$), then $c_{ij} = 1$. To solve the conflicts, we may introduce stitches to split the original patterns into several touching parts, which may leads to overlapping issues and thus reducing yields. There are lots of candidate stitch insertion positions, and we only choose some stitches from those candidates. The objective function is to minimize the number of conflicts and stitches at the same time.

- **Integer Linear Programming**: Equation (1) is nonlinear and difficult to solve. A widely used method to solve this problem is Integer linear programming (ILP) [2, 16–18], which converts this problem into linear programming by binary encoding of vertex colors and replaces the color comparison operations with a set of inequality constraints. ILP can be easily extended to solve different types of coloring problem, including TPLD and QPLD. Our framework is based on the theory proposed in [2]. We use Gurobi [19], Lemon [20] and CBC [21] as ILP solvers for OpenMPL.
- **Linear Programming**: Compared with integer linear programming, linear programming (LP) [5] constructs the model without strict constraints on integers and relaxes the problem to non-integer solution space. Then LP uses iterative rounding methods to converge the results into integer solution space. In our work, LP is solved by calling Gurobi API [19].
- **Semidefinite Programming**: The discrete integer programming solving process of Equation (1) is NP-hard. As shown in [2, 7, 8], We can use semidefinite programming to relax the problem and therefore can be solved in polynomial time. CSDP [22] and OpenBLAS [23] are used in our work as the semidefinite programming solvers.
- **Backtracking**: Backtracking is a DFS fashion algorithm, used to find solutions with some constraints in the whole solution space. We also provide the backtracking solver in our framework. Though backtracking is widely used, its performance is unsatisfactory for complicated graphs.
- **Dancing Links**: Different from original dancing links and algorithm X, dancing link solver [10] concludes conflicts earlier through BFS style traversal, treating this problem as an exact cover problem. In OpenMPL, this solver is still an independent component and needs more optimization.

Users of our framework can specify which algorithm is adopted and the number of available colors.

2.2 Layout Simplification

Layout graph simplification techniques can be used to reduce the graph size and therefore reduce the computational complexity. We only need to deal with the smaller graph without affecting the final result if the graph is simplified. Four simplification levels are supported in our framework with [24]:

- Level 0: No simplification. Use the whole original layout graph as the input.
- Level 1: `HIDE_SMALL_DEGREE`. Temporarily remove the nodes whose degree is smaller than the number of color variables [2, 5, 13]. After the coloring of the remaining nodes in the graph, we will recover these nodes back to the graph and assign them suitable colors sequentially. We conduct the remove process iteratively until all the low-degree nodes are removed.
- Level 2: `MERGE_SUBK4`. Detect and merge four-clique structures [5]. Obviously, 4-clique is not 3-colorable and cannot be solved through simple algorithms, which may need design correction. This method can also be used in native conflict detections.
- Level 3: `BICONNECTED_COMPONENT`. Biconnected component decomposition [5, 12, 14, 15]. After partitioning the original layout graph into several independent components, we can conduct the layout decomposition algorithms on each component in parallel. After coloring, we then compose the components together to obtain final results.

It is necessary to keep popping order during node recovery, so we store all the temporarily-removed nodes in the stack during simplification. Then we add the nodes back according to the stack ordering in the recovery process. Since each component is processed

independently in parallel, it is possible that the shared nodes in different components may have different colors. Therefore, color rotation is also implemented in our framework to tackle that.

2.3 Additional Features

Besides the algorithms and simplification levels, additional features are supported.

- OpenMPL supports **multithreading** operation and users can specify the number of threads. The graph components are solved in parallel and the layout decomposition algorithms also support multithreading computations.
- We can identify all the possible positions for stitches through pattern projections in **stitch insertion**. The insertion strategies for different MPLD problems are distinct from each other. The TPLD and QPLD are more complex while DPLD is relatively simpler. Currently, a prototype version of stitch insertion has been implemented and a comprehensive one will be integrated into the framework soon.

In practice, a pattern in the layout may be polygon or rectangle. Consequently, the storages may vary from each other. For polygonal inputs, to simplify the storage structure design and save space, OpenMPL firstly decomposes the polygons to rectangles. After reading the whole input file, DFS is utilized to find connected components and re-union rectangles into polygons. For rectangle circuits, we directly store these patterns without further operations. Despite this, to guarantee the performance, users can still specify the shape, `POLYGON` or `RECTANGLE`, to avoid unnecessary calculations. We use Boost [25] as the basic graphics library.

3 EXPERIMENTAL RESULTS

We conduct a set of experiments on ISCAS_sim and ISCAS_total benchmarks which are widely used in previous works.

Four layout decomposition algorithms, ILP, LP, SDP and Backtracking are tested on both QPLD and TPLD. For QPLD, we set the coloring distance as 160nm and the thread number is 8. As to TPLD, the coloring distance is 120nm and the thread number is 8. The graph simplification level is 3 (`BICONNECTED_COMPONENT`). The CPU time (s) of TPLD and QPLD experiments are in Figure 3 and Figure 4, respectively. The CPU time contains the time of the file I/O operations, graph simplifications, layout decompositions and graph recoveries. Since CPU times vary widely for different circuits, the results in the charts have been taken the logarithm of 2. In Figure 3 and Figure 4, some CPU times are negative because the time consumed are much less than 2 seconds. For the cases whose CPU times are more than 3600 seconds, we directly terminated the computations and the CPU times are not listed. The numbers of conflicts for TPLD and QPLD are in Figure 5 and Figure 6. Because we have already inserted candidate stitches into the layout graphs to eliminates some conflicts, the final conflict numbers in the charts mean the unsolved conflicts during to the design defects. The conflict numbers of the cases which need very high CPU times are also not shown in the charts.

4 CONCLUSION AND FUTURE WORK

OpenMPL is a general framework which provides unified interfaces for layout decomposition algorithms and graph simplification speed-up techniques. Multithreading is also supported in our framework. All of these are customizable and users can switch between these options freely.

This version of OpenMPL uses the preprocessed circuits as the inputs in which the candidates stitches have already been identified. We have already developed a simple version of stitches insertion program, and in near future a comprehensive one will be integrated into OpenMPL. The alpha version of dancing link solver has been integrated into the system and it is being actively optimized for further improvement.

REFERENCES

[1] "OpenMPL," https://github.com/limbo018/OpenMPL.
[2] B. Yu, K. Yuan, D. Ding, and D. Z. Pan, "Layout decomposition for triple patterning lithography," *IEEE TCAD*, vol. 34, no. 3, pp. 433–446, March 2015.
[3] H. Zhang, Y. Du, M. D. F. Wong, and R. O. Topaloglu, "Characterization and decomposition of self-aligned quadruple patterning friendly layout," in *Proc. SPIE*, vol. 8326, 2012.
[4] B. Yu, J.-R. Gao, and D. Z. Pan, "Triple patterning lithography (TPL) layout decomposition using end-cutting," in *Proc. SPIE*, vol. 8684, 2013.
[5] Y. Lin, X. Xu, B. Yu, R. Baldick, and D. Z. Pan, "Triple/quadruple patterning layout decomposition via novel linear programming and iterative rounding," in *SPIE Advanced Lithography*, vol. 9781, 2016.
[6] B. Yu, Y.-H. Lin, G. Luk-Pat, D. Ding, K. Lucas, and D. Z. Pan, "A high-performance triple patterning layout decomposer with balanced density," in *Proc. ICCAD*, 2013, pp. 163–169.
[7] B. Yu and D. Z. Pan, "Layout decomposition for quadruple patterning lithography and beyond," in *Proc. DAC*, 2014, pp. 53:1–53:6.
[8] T. Matsui, Y. Kohira, C. Kodama, and A. Takahashi, "Positive semidefinite relaxation and approximation algorithm for triple patterning lithography," in *Proc. ISAAC*, 2014, pp. 365–375.
[9] Y. Kohira, T. Matsui, Y. Yokoyama, C. Kodama, A. Takahashi, S. Nojima, and S. Tanaka, "Fast mask assignment using positive semidefinite relaxation in LELE-CUT triple patterning lithography," in *Proc. ASPDAC*, 2015, pp. 665–670.
[10] H.-Y. Chang and I. H.-R. Jiang, "Multiple patterning layout decomposition considering complex coloring rules," in *Proc. DAC*, 2016, pp. 40:1–40:6.
[11] J.-S. Yang, K. Lu, M. Cho, K. Yuan, and D. Z. Pan, "A new graph-theoretic, multi-objective layout decomposition framework for double patterning lithography," in *Proc. ASPDAC*, 2010, pp. 637–644.
[12] S.-Y. Fang, Y.-W. Chang, and W.-Y. Chen, "A novel layout decomposition algorithm for triple patterning lithography," *IEEE TCAD*, vol. 33, no. 3, pp. 397–408, March 2014.
[13] J. Kuang and E. F. Y. Young, "An efficient layout decomposition approach for triple patterning lithography," in *Proc. DAC*, 2013, pp. 69:1–69:6.
[14] A. B. Kahng, C.-H. Park, X. Xu, and H. Yao, "Layout decomposition for double patterning lithography," in *Proc. ICCAD*, 2008, pp. 465–472.
[15] K. Yuan, J.-S. Yang, and D. Z. Pan, "Double patterning layout decomposition for simultaneous conflict and stitch minimization," in *Proc. ISPD*, 2009, pp. 107–114.
[16] Y. Xu and C. Chu, "GREMA: graph reduction based efficient mask assignment for double patterning technology," in *Proc. ICCAD*, 2009, pp. 601–606.
[17] A. B. Kahng, C.-H. Park, X. Xu, and H. Yao, "Layout decomposition approaches for double patterning lithography," *IEEE TCAD*, vol. 29, pp. 939–952, June 2010.
[18] K. Yuan, J.-S. Yang, and D. Z. Pan, "Double patterning layout decomposition for simultaneous conflict and stitch minimization," *IEEE TCAD*, vol. 29, no. 2, pp. 185–196, Feb. 2010.
[19] Gurobi Optimization Inc., "Gurobi optimizer reference manual," http://www.gurobi.com, 2016.
[20] "LEMON," http://lemon.cs.elte.hu/trac/lemon.
[21] "CBC," http://www.coin-or.org/projects/Cbc.xml.
[22] B. Borchers, "CSDP, a C library for semidefinite programming," *Optimization Methods and Software*, vol. 11, pp. 613–623, 1999.
[23] "OpenBLAS," http://www.openblas.net/.
[24] "Limbo," http://yibolin.com/Limbo/docs/html/index.html.
[25] "Boost C++ Library," http://www.boost.org.

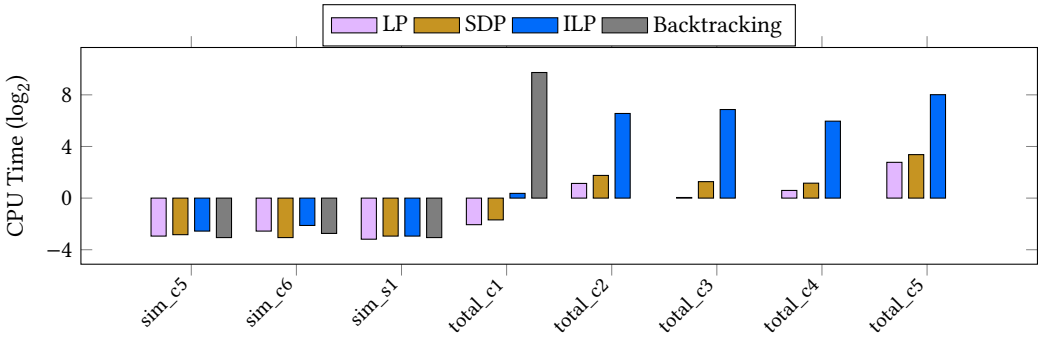

Figure 3: The CPU times of TPLD experiments.

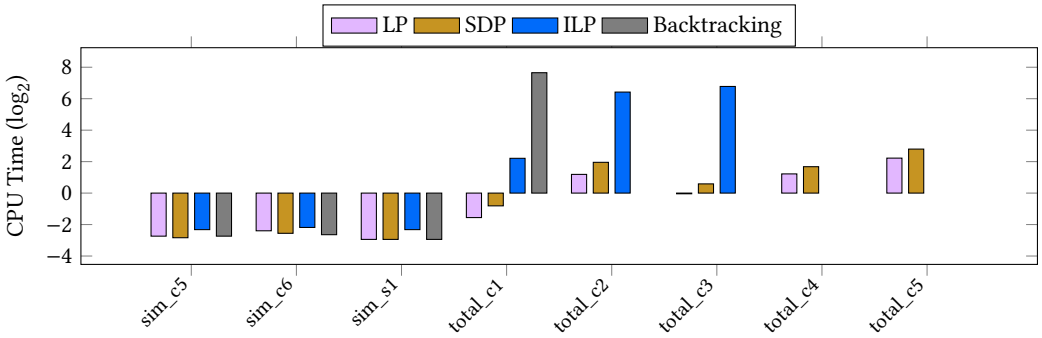

Figure 4: The CPU times of QPLD experiments.

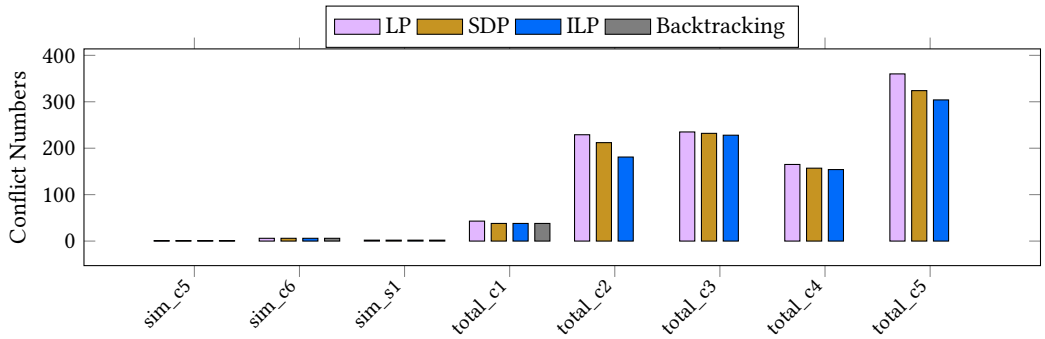

Figure 5: The numbers of conflicts of TPLD experiments.

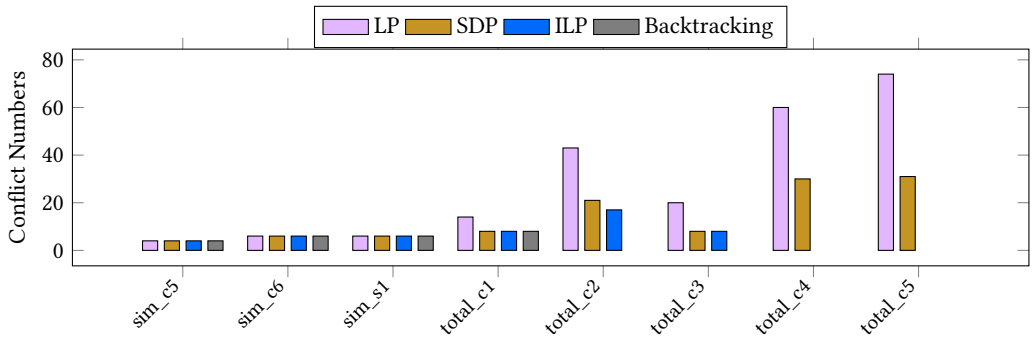

Figure 6: The numbers of conflicts of QPLD experiments.