Model-based OPC Extension in OpenILT

Su Zheng[1], Gang Xiao[2], Ge Yan[2], Meng Dong[2], Yao Li[2], Hong Chen[2], Yuzhe Ma[3], Bei Yu[1], Martin Wong[4]
[1] The Chinese University of Hong Kong, [2] Shenzhen GWX Technology Co., Ltd.,
[3] Hong Kong University of Science and Technology (Guangzhou), [4] Hong Kong Baptist University
[1] {szheng22, byu}@cse.cuhk.edu.hk, [2] {xiaogang, yange, dongmeng, liyao, ch}@gwxeda.com

***Abstract*—Optical proximity correction (OPC) is a technique to improve the accuracy of pattern transfer from the mask to the wafer in optical lithography. Model-based OPC (MB-OPC) uses mathematical models to simulate the image formation process and adjust the mask layout accordingly. In this paper, we extend the open-source computational lithography library OpenILT to support MB-OPC. The extension provides a flexible and modular framework for implementing OPC algorithms with GPU acceleration for large-scale layouts. We demonstrate the performance and scalability of the library on different mask patterns. The experimental results show that our method can achieve more than 5 times speedup over the CPU-based MB-OPC method, while maintaining the same correction accuracy and quality. Our MB-OPC extension can provide a powerful baseline for future research on OPC.**

***Index Terms*—Optical Proximity Correction, Computational Lithography, GPU Acceleration**

I. Introduction

Lithography is a key technology for the fabrication of integrated circuits and nanophotonic devices. However, as the feature sizes of these devices approach the diffraction limit of light, lithography faces significant challenges in achieving high resolution and pattern fidelity. Optical proximity correction (OPC) [1]–[9] is a technique to compensate for the optical and process distortions that occur during the pattern transfer from the mask to the wafer. It modifies the mask layout by adding or subtracting small features, such as serifs or subresolution assist features, to improve the printability and accuracy of the desired pattern.

Among the various OPC methods, model-based OPC (MB-OPC) [10], [11] is a widely used one in the industry, as it can handle complex and arbitrary mask patterns. MB-OPC relies on mathematical models to simulate the image formation process and optimize the mask layout accordingly. The models include the optical projection model, which describes the light propagation and interference in the optical system, and the resist model, which describes the chemical and physical reactions in the photoresist layer. Fig. 1 summarizes the model-based OPC flow, which includes the segmentation of a layout, lithography simulation of the mask, and movement of the segments. The accuracy and efficiency of MB-OPC depend largely on the quality and complexity of the lithography simulation and movement strategy.

However, developing and calibrating accurate models for MB-OPC is not a trivial task, as it requires extensive experimental data and computational resources. Moreover, the models may need to be updated frequently to adapt to the changes in the lithography process and equipment. Therefore, there is a need for a flexible and modular framework for implementing and testing different OPC models and algorithms.

Authors acknowledge Shenzhen Science and Technology Program (KQTD20230301092840008) for research fund support.

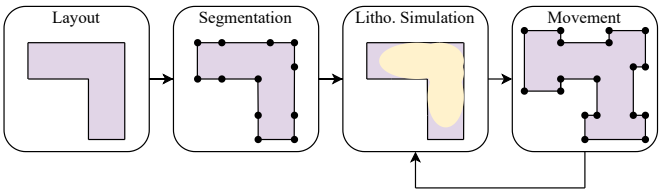


Fig. 1 Overview of a typical MB-OPC flow.

MB-OPC is computationally intensive and requires a large amount of CPU resources and runtime, especially for large-scale layouts with trillions of shapes. However, the MB-OPC for large-scale layout is not extensively explored in the literature, and most of the existing methods are based on serial or parallel CPU implementations. In this paper, we propose a novel method to significantly accelerate large-scale MB-OPC via GPU acceleration. By dividing the large-scale layout into smaller sub-regions that can fit into the GPU memory, our method leverages the massive parallelism and high memory bandwidth of GPUs to perform the MB-OPC.

OpenILT [12] aims to facilitate computational lithography research. With effective acceleration, convenient interfaces, and comprehensive evaluation, OpenILT can make computational lithography research easier. However, OpenILT mainly focuses on inverse lithography technique (ILT) [13]–[23] and does not support model-based OPC, which limits its compatibility and capability in related research and applications. Furthermore, OpenILT cannot deal with large-scale lithography simulation and thus it cannot be applied to large-scale OPC problems. Thus, extending OpenILT to support large-scale lithography simulation and MB-OPC can make contributions to the field of computational lithography.

In this paper, we present an MB-OPC extension for OpenILT that can handle large-scale layouts. The major contributions of our work are summarized as follows.

- Our MB-OPC extension to OpenILT enables a convenient and modular approach to the implementation of OPC techniques.
- By partitioning a layout into patches, we enable the lithography simulation and OPC of large-scale layout in OpenILT.

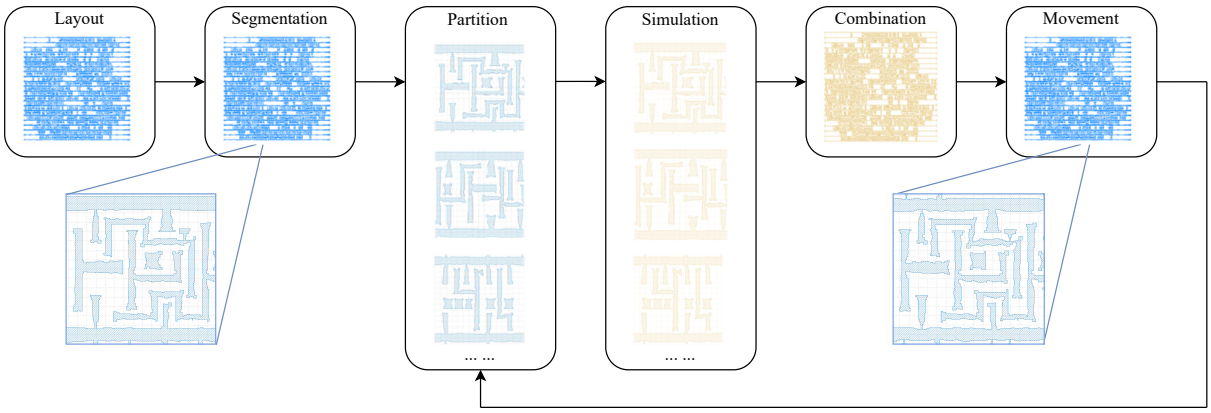


Fig. 2 Overview of the proposed model-based OPC flow.

- We leverage GPU resources to accelerate the OPC computation, achieving significant speedup over the CPU-based MB-OPC method, while maintaining the same correction accuracy and quality.
- We present the efficiency and scalability of the library on large-scale mask patterns. Our MB-OPC extension can serve as a robust platform for future OPC research.

II. Preliminaries

Fig. 1 shows a typical MB-OPC flow. Given a layout consisting of multiple polygons, we divide each polygons into segments and make them movable. The segments to be optimized are transformed into the mask image. The lithography model simulates the process that transfers the mask image to the printed image. According to the simulation result, we move the segments to improve the quality of the printed patterns. Finally, the optimized mask is obtained after specific iterations of optimization.

A. Lithography Simulation Model

The lithography simulation process includes the optical projection model, which describes the light propagation and interference in the optical system, and the resist model, which describes the chemical and physical reactions in the photoresist layer. We apply the Hopkins diffraction model [24] to estimate how the projection behaves. We compute the aerial image $\boldsymbol{I}$ by applying a series of optical kernels $\boldsymbol{H}$ to the mask $\boldsymbol{M}$. In this paper, the computation is given by,

$$\boldsymbol{I}(x,y) = \sum_{k=1}^{N_h} w_k \left| \boldsymbol{M}(x,y) \otimes \boldsymbol{h}_k(x,y) \right|^2, \tag{1}$$

where "$\otimes$" is the convolution operation. $N_h = 24$ is the number of optical kernels in our implementation. The notation $\boldsymbol{h}_k$ represents the k-th optical kernel in $\boldsymbol{H}$. The weight of this kernel is given by w_k.

Next, the resist model is applied to the aerial image $\boldsymbol{I}$. The printed image $\boldsymbol{Z}$ is computed by the following function:

$$\boldsymbol{Z}(x,y) = \sigma_Z(\boldsymbol{I}(x,y)) = \frac{1}{1+\exp\left(-\theta_Z(\boldsymbol{I}(x,y) - I_{th})\right)}, \tag{2}$$

where θ_Z controls the steepness of the sigmoid function. The intensity threshold I_{th} indicates the exposure level. For evaluation, both $\boldsymbol{M}$ and $\boldsymbol{Z}$ should be binarized with a threshold of 0.5.

B. Evaluation Metrics

Square L_2 Error. We use the square L_2 loss $||\boldsymbol{Z}_{nom} - \boldsymbol{Z}_t||_2^2$ to estimate the error of the printed image, where $\boldsymbol{Z}_t$ is the target image and $\boldsymbol{Z}_{nom}$ is the printed image that represents the image printed via nominal lithography process condition.

Edge placement error (EPE). EPE evaluates the contour difference between the target design $\boldsymbol{Z}_t$ and the image $\boldsymbol{Z}_{nom}$. For MB-OPC, we count the EPEs according to the distance between each segment in the target image and its corresponding contour in the printed image. If the distance is larger than a threshold, we regard it as an EPE.

Process Variation Band (PVB). Different lithography conditions can produce different printed images. Thus, we need to compare the distance of printed images between varying conditions to show the robustness of the optimized mask. After simulating the printed images under the maximum and minimum lithography conditions, we get $\boldsymbol{Z}_{max}$ and $\boldsymbol{Z}_{min}$, and the PVB is calculated by $||\boldsymbol{Z}_{max} - \boldsymbol{Z}_{min}||_2^2$.

C. GPU Acceleration for Lithography Simulation

A convolution operation in eq. (1) involves two matrices that have large sizes (e.g. 2048×2048), which can be very time-consuming. Thus, the convolution operation is usually implemented by fast Fourier transformation (FFT) to improve efficiency, which can be formulated as:

$$\boldsymbol{h}_k \otimes \boldsymbol{M} = \texttt{IFFT}(\texttt{FFT}(\boldsymbol{h}_k) \odot \texttt{FFT}(\boldsymbol{M})), \tag{3}$$

where `FFT` and `IFFT` represent the FFT and inverse FFT operations, respectively. The notation $\odot$ represents pointwise multiplication. Computing the optical projection with eq. (1) can significantly reduce the time complexity, since using `FFT` and `IFFT` for $N \times N$ images requires $O(N^2 \log N)$, whereas directly computing the convolution costs $O(N^4)$. Furthermore, the computation can be further accelerated by

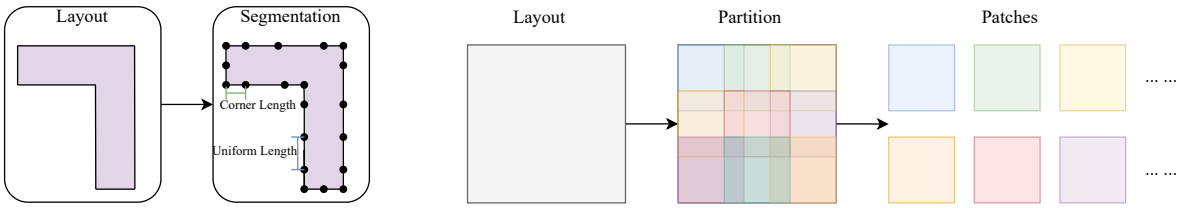


Fig. 3 Illustration of steps (a) segmentation, (b) partition.

removing the high-frequency components from the optical kernels and employing GPU acceleration.

Based on PyTorch [25] and OpenILT, we can implement the reference lithography simulation model as a PyTorch module, which can be used like a DNN layer. The GPU-based fast Fourier transform (FFT) can boost the speed of lithography simulation. Similar acceleration schemes are adopted in state-of-the-art works like [26]–[29].

III. Method

A. Overview

Fig. 2 shows the overview of our MB-OPC flow, which consists of the following steps.

- **Segmentation**. This step divides each polygon in a given layout into a set of segments, which can represent the whole mask. The patterns in the layout can be corrected by moving the segments. The purpose of segmentation is to create the variables for mask optimization.
- **Partition**. Since the lithography simulator in OpenILT can only support the simulation of patches that are smaller than $2 \times 2\mu m^2$, we need to enable the lithography simulation of a large-scale layout by partitioning into smaller patches and combine the results to form the printed image. The patches are typically square or rectangular, and their size and shape depend on the resolution and memory of the GPU. It can parallelize the simulation process and speed up the computation.
- **Simulation**. This step uses the lithography model to simulate the image formation process on the wafer. The GPU-based simulation process parallelly predict the printed image of each patch, which can be combined and compared with the target layout.
- **Combination**. This step combines the simulated patches into the whole printed image, and calculates the error metrics such as edge placement error (EPE). The error metrics measure the deviation of the printed image from the target layout, and they can be used to guide the correction process. The purpose of the combination step is to evaluate the overall performance of the mask and identify the areas that need improvement.
- **Movement**. This step moves the segments to improve the quality of the printed image, by minimizing the error metrics. The movement is carried out iteratively until a satisfactory solution is found. In each iteration, the partition, simulation, and combination steps should also be done. The purpose of movement is to correct the mask pattern and achieve the best possible match with the target layout.

B. Segmentation

The segmentation process takes two parameters as input: the corner length and the uniform length. These parameters determine the sizes of the segments that represent the mask pattern.

As shown in Fig. 3(a), the corner length parameter decides the length of segments around corners of polygons. This parameter is usually smaller than the uniform length parameter, because corners are more sensitive to image errors and need more correction. The uniform length parameter decides the length of other segments that are not around corners. This parameter is usually larger than the corner length parameter, because these segments are less sensitive to image errors and need less correction.

The process starts from the corners of the polygons, and creates segments with the corner length parameter along the edges. Then, the process fills the gaps between the corner segments with segments with the uniform length parameter. The process repeats for all the polygons in the layout, until the whole mask pattern is represented by segments. After that, we can correct the mask by moving the segments according to the simulation result.

C. Partition

To simulate a large-scale layout with the lithography simulator in OpenILT, which has a limit of $2\times2\mu m^2$ patches, we have to split the layout into smaller patches, simulate each patch, and then merge the simulated images to get the final printed image. The partition process takes the whole mask image as input, and divides it into smaller patches with some overlaps, as done in [30]. The patches are usually square or rectangular, and their size and shape depend on the resolution and memory of the GPU. The patches should be large enough to capture the optical effects of the mask pattern, but small enough to fit into the GPU memory and avoid unnecessary computation. The patches should also have some overlaps with each other, to ensure the continuity and accuracy of the simulated image.

Fig. 3(b) illustrates the partition process. The process starts from the top-left corner of the mask image, and creates a patch with the specified size and overlap. Next, the process

TABLE I MB-OPC Results on Different Testcases

Layout	Size	Tiles	EPE	L2	PVB	Runtime@GPU	Runtime@CPU
gcd	$30 \times 30\mu m^2$	1	24657	595317	149969	150 s	801 s
aes	$250 \times 250\mu m^2$	144	2633481	62804952	16102282	18,612 s	>24 hours
dynamicnode	$246 \times 246\mu m^2$	144	2326174	58335791	14815929	19,494 s	>24 hours

moves to the right by one patch width minus the overlap, and creates another patch. The process repeats until it reaches the right edge of the mask image. After that, the process moves down by one patch height minus the overlap, and repeats the same procedure for the next row of patches. The process repeats this until it covers the whole mask image with patches.

D. Simulation

The simulation process uses the lithography model to calculate the printed image intensity for each patch. To accelerate the simulation, we group the patches by batches and simulate them on GPU, performing fast and efficient calculations for large amounts of data.

The simulation process can use different batch sizes to balance the trade-off between speed and resource requirements. The batch size is the number of patches that are simulated together on the GPU. A larger batch size can increase the GPU utilization and reduce the data transfer time and computation latency, but it can also increase the memory requirement. A smaller batch size can decrease the memory requirement, but it can also decrease the GPU utilization and increase the data transfer time and computation latency. The optimal batch size depends on the patch size, the GPU memory, and the settings of the lithography model.

E. Combination

The combination process takes the simulated patches as input, and combines them into the printed image of the whole mask. It uses a stitching method to merge the simulated patches into the printed image. The stitching method aligns the patches according to their original positions in the mask image, and averages the overlapping regions to avoid discontinuities or artifacts. The stitched printed image shows the predicted shape and intensity of the features on the wafer, and the error metrics can be calculated to show the deviation of the printed image from the target layout.

After stitching, the combination process also calculates the error metrics for the printed image, such as edge placement error (EPE). The error metrics can be calculated for each segment, and they can be used to guide the correction process. Finally, the combination process outputs the printed image and the error metrics for the whole mask image, which can guide the movement of the segments.

F. Movement

The movement process takes the printed image and the error metrics as input, and moves the segments to improve the quality of the printed image. It uses an optimization method to minimize the error metrics, such as edge placement error (EPE). Specifically, we move the segments which have EPEs. If the segment has an inner EPE, which means that the corresponding contour is inside the target pattern, we move the segment outwards by a configurable step size. On the contrary, if the segment has an outer EPE, which means that the corresponding contour is outside the target pattern, we move the segment inwards by the step size. The movement process iterates until a satisfactory solution is found, or a predefined termination criterion is met. Note that each iteration also involves the partition, simulation, and combination steps.

IV. Experiments

A. Experimental Settings

In this experiment, we apply the proposed model-based optical proximity correction (MB-OPC) method to three designs: gcd, aes, and dynamicnodes. These designs are obtained from the OpenROAD tool [31], which is an open-source, autonomous, RTL-to-GDSII flow for rapid integrated circuit implementation. The tested technology node is 45 nm, which is a challenging lithography scenario that requires accurate and robust OPC techniques. Before applying MB-OPC, we crop the metal layer of every design into $30 \times 30\mu m^2$ tiles to enable parallel processing on GPU. The hardware platform for our experiments is equipped with an NVIDIA RTX 2060 12G GPU, an Intel i7-10710u CPU, and 64G DDR4 memory.

B. MB-OPC Results on Tested Designs

The MB-OPC results on the testcases gcd, aes, and dynamicnodes are presented in TABLE I. Our MB-OPC extension can finish within 6 hours on every design, achieving a speedup of $5.3\times$ when compared to the CPU version.

The GPU-based lithography simulation makes a major contribution to the significant speedup. The GPU acceleration for lithography simulation is faster than CPU because GPU has a massively parallel architecture that can handle thousands of threads simultaneously, while CPU has a limited number of cores that can process a few threads at a time. GPU can also perform fast matrix operations and floating-point calculations, which are essential for MB-OPC. Moreover, GPU can leverage the power of CUDA and PyTorch, which enables GPU-accelerated computational lithography and mask data preparation. Therefore, GPU acceleration is a powerful technique that can significantly improve the efficiency and quality of MB-OPC for complex and challenging lithography scenarios. Our extension can serve as a baseline for efficient GPU-accelerated MB-OPC.

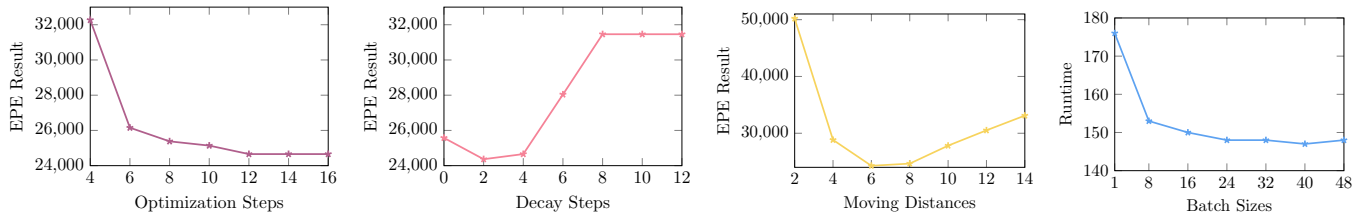


Fig. 4 Results under different configurations. Note that when we vary a hyper-parameter, others are set to the default values. The subfigures compare different (a) optimization steps; (b) decay steps; (c) moving distances; (d) batch sizes.

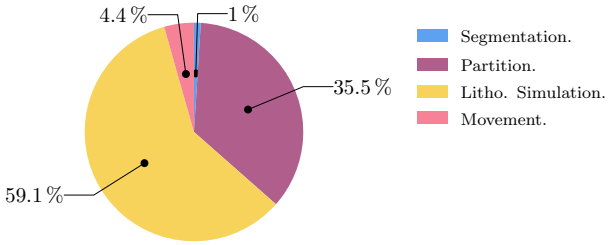


Fig. 5 Runtime analysis of the MB-OPC flow.

C. Impacts of Different Hyper-parameters

By default, the MB-OPC process takes 12 steps with a moving distance of 8 nm. The moving distance is decayed to 4 nm after the 4th iteration. The GPU simulation uses a batch size of 16. In this section, we investigate the impacts of these hyper-parameters by testing the performance under different configurations on the `gcd` testcase.

Fig. 4 presents the comparison of EPEs or runtimes under different configurations. Fig. 4(a) shows the relationship between the number of optimization steps and the EPE result. The figure shows that the EPE result decreases as the number of optimization steps increases, meaning that the mask becomes more optimized. However, after 12 optimization steps, the EPE result does not change significantly, suggesting that further optimization has no benefit. Therefore, an optimization step of 12 is proper for this layout and others that have similar complexity.

Fig. 4(b) shows the relationship between the number of decay steps and the EPE result. It indicates that the result worsens as the number of decay steps increases, meaning that the optimization algorithm becomes less effective. However, a decay step of 2 or 4 seems to achieve a good result, suggesting that a small decay step is beneficial for the MB-OPC performance. On the other hand, if we set the decay step to 0, which means that the moving distance is reduced before optimization starts, the result is unsatisfactory, implying that the moving distance is too small for the optimization.

Fig. 4(c) demonstrates the relationship between the moving distance and the EPE result. The figure shows that the performance is optimal when the moving distance is 6 or 8, meaning that the solution can explore the search space efficiently. However, if the moving distance is too small or too large, the performance deteriorates, implying that the solution either gets stuck in a local optimum or moves too far away from the optimal region.

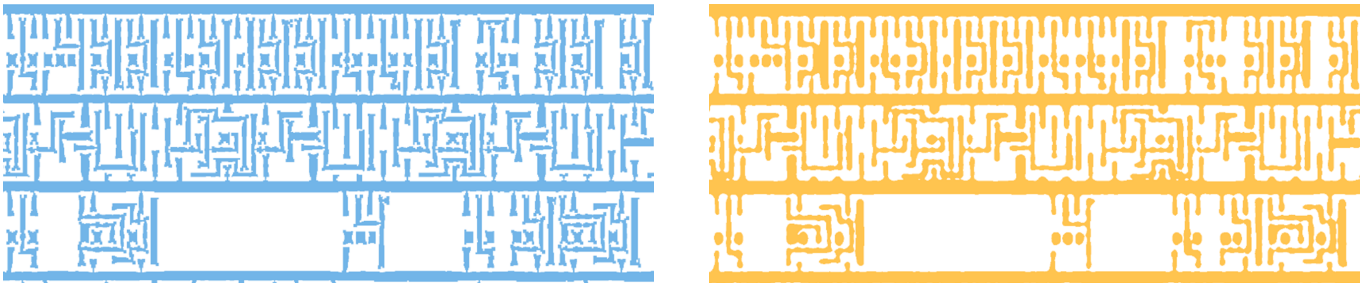
Fig. 4(d) shows the relationship between the batch size and the runtime for our MB-OPC method. It indicates that the runtime decreases as the batch size increases, meaning that we can process more data in less time. However, after the batch size reaches 16, the runtime does not change significantly, suggesting that further increasing the batch size has little benefit for the runtime. Therefore, a batch size of 16 or slightly higher is optimal for this testcase.

D. Runtime Analysis

Fig. 5 shows a pie chart that illustrates the percentage of runtime spent on different components of an optimization algorithm for a photonic device design problem. The pie chart has four slices: segmentation, partition, lithography simulation, and movement. The largest slices are partition and lithography simulation, which take up 35.5% and 59.1% of the runtime, respectively. On the contrary, segmentation and movement only consume less than 6% of the runtime. This experiment indicates that most of the runtime is spent on partition and lithography simulation (including combination), which together account for 94.6% of the runtime. Segmentation and movement only take a very small portion of the runtime, which suggests that they are not the bottleneck of the optimization algorithm. In future work, faster partition and lithography simulation can be promising directions that can significantly boost the speed of MB-OPC. An example of the MB-OPC result is shown in Fig. 6.

V. Conclusion and Future Work

This paper presents an MB-OPC extension of OpenILT, an open-source library for computational lithography, to enable model-based optical proximity correction. It breaks the layout size limitation of the lithography simulation in OpenILT and offers a flexible and modular platform for developing OPC algorithms for large-scale mask layouts. It also leverages GPU computing to accelerate the OPC process. Experiments show that our method can achieve a significant speedup of more than 5 times. Our MB-OPC extension can serve as a robust baseline for future OPC research and facilitate the future development of this field.



(a) (b)

Fig. 6 An MB-OPC example, where (a) is the optimized mask and (b) is the simulation result.

In future works, researcher can easily enhance MB-OPC performance by improve our flow. Some possible research directions for MB-OPC are shown as follows.

- Developing data-driven approaches that use machine learning techniques to predict the fragment displacements or generate the simulated images, which could reduce the computational cost and improve the accuracy of MB-OPC.
- Proposing novel methods to enable more efficient partition, simulation, and optimization. For example, the movement of each segment can consider a wider environment around it.
- Integrating mask rule checking and correction algorithms that can legalize the optimized patterns according to the mask rules, which could ensure the manufacturability and quality of the final layout.

REFERENCES

[1] J.-S. Park, C.-H. Park, S.-U. Rhie, Y.-H. Kim, M.-H. Yoo, J.-T. Kong, H.-W. Kim, and S.-I. Yoo, "An efficient rule-based OPC approach using a DRC tool for 0.18/spl mu/m ASIC," in *Proc. ISQED*, 2000, pp. 81–85.

[2] A. Awad, A. Takahashi, S. Tanaka, and C. Kodama, "A fast process variation and pattern fidelity aware mask optimization algorithm," in *Proc. ICCAD*, 2014, pp. 238–245.

[3] J. Ou, B. Yu, J.-R. Gao, D. Z. Pan, M. Preil, and A. Latypov, "Directed self-assembly based cut mask optimization for unidirectional design," in *Proc. GLSVLSI*, 2015, pp. 83–86.

[4] Y. Ma, X. Zeng, and B. Yu, "Methodologies for layout decomposition and mask optimization: A systematic review," in *Proc. VLSI-SoC*, 2017.

[5] Y. Ma, J.-R. Gao, J. Kuang, J. Miao, and B. Yu, "A unified framework for simultaneous layout decomposition and mask optimization," in *Proc. ICCAD*, 2017, pp. 81–88.

[6] W. Zhong, S. Hu, Y. Ma, H. Yang, X. Ma, and B. Yu, "Deep learning-driven simultaneous layout decomposition and mask optimization," in *Proc. DAC*, 2020.

[7] H. Yang, W. Zhong, Y. Ma, H. Geng, R. Chen, W. Chen, and B. Yu, "VLSI mask optimization: From shallow to deep learning," *Integration, the VLSI Journal*, vol. 77, pp. 96–103, 2021.

[8] W. Zhao, X. Yao, Z. Yu, G. Chen, Y. Ma, B. Yu, and M. D. Wong, "AdaOPC: A self-adaptive mask optimization framework for real design patterns," in *Proc. ICCAD*, 2022.

[9] Z. Yu, P. Liao, Y. Ma, B. Yu, and M. D. Wong, "CTM-SRAF: Continuous transmission mask-based constraint-aware sub resolution assist feature generation," *IEEE TCAD*, 2023.

[10] J. Kuang, W.-K. Chow, and E. F. Y. Young, "A robust approach for process variation aware mask optimization," in *Proc. DATE*, 2015, pp. 1591–1594.

[11] H. T. Vu, S. Kim, J. Word, and L. Y. Cai, "A novel processing platform for post tape out flows," in *Proc. SPIE*, vol. 10587, 2018, pp. 219–224.

[12] S. Zheng, B. Yu, and M. Wong, "OpenILT: An open source inverse lithography technique framework," in *Proc. ASICON*, 2023.

[13] Y. Liu, D. Abrams, L. Pang, and A. Moore, "Inverse lithography technology principles in practice: Unintuitive patterns," in *BACUS Symposium on Photomask Technology*, vol. 5992, 2005, pp. 886–893.

[14] J.-R. Gao, X. Xu, B. Yu, and D. Z. Pan, "MOSAIC: Mask optimizing solution with process window aware inverse correction," in *Proc. DAC*, 2014.

[15] Z. Yu, G. Chen, Y. Ma, and B. Yu, "A GPU-enabled Level Set Method for Mask Optimization," in *Proc. DATE*, 2021, pp. 1835–1838.

[16] ——, "A GPU-enabled level-set method for mask optimization," *IEEE TCAD*, vol. 42, no. 2, pp. 594–605, 2022.

[17] H. Yang, S. Li, Y. Ma, B. Yu, and E. F. Young, "GAN-OPC: Mask optimization with lithography-guided generative adversarial nets," in *Proc. DAC*, 2018.

[18] H. Yang, S. Li, Z. Deng, Y. Ma, B. Yu, and E. F. Young, "GAN-OPC: Mask optimization with lithography-guided generative adversarial nets," *IEEE TCAD*, vol. 39, no. 10, pp. 2822–2834, 2019.

[19] B. Jiang, L. Liu, Y. Ma, H. Zhang, B. Yu, and E. F. Young, "Neural-ILT: Migrating ILT to neural networks for mask printability and complexity co-optimization," in *Proc. ICCAD*, 2020.

[20] B. Jiang, L. Liu, Y. Ma, B. Yu, and E. F. Young, "Neural-ILT 2.0: Migrating ILT to Domain-Specific and Multitask-Enabled Neural Network," *IEEE TCAD*, vol. 41, no. 8, pp. 2671–2684, 2021.

[21] G. Chen, W. Chen, Y. Ma, H. Yang, and B. Yu, "DAMO: Deep agile mask optimization for full chip scale," in *Proc. ICCAD*, 2020.

[22] G. Chen, W. Chen, Q. Sun, Y. Ma, H. Yang, and B. Yu, "DAMO: Deep agile mask optimization for full-chip scale," *IEEE TCAD*, vol. 41, no. 9, pp. 3118–3131, 2022.

[23] G. Chen, Z. Yu, H. Liu, Y. Ma, and B. Yu, "DevelSet: Deep neural level set for instant mask optimization," in *Proc. ICCAD*, 2021.

[24] H. H. Hopkins, "The concept of partial coherence in optics," in *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences*, vol. 208, no. 1093. The Royal Society London, 1951, pp. 263–277.

[25] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Proc. NeurIPS*, vol. 32, 2019.

[26] H. Yang, Z. Li, K. Sastry, S. Mukhopadhyay, M. Kilgard, A. Anandkumar, B. Khailany, V. Singh, and H. Ren, "Generic lithography modeling with dual-band optics-inspired neural networks," in *Proc. DAC*, 2022, pp. 973–978.

[27] S. Sun, F. Yang, B. Yu, L. Shang, and X. Zeng, "Efficient ILT via Multi-level Lithography Simulation," in *Proc. DAC*, 2023.

[28] S. Zheng, H. Yang, B. Zhu, B. Yu, and M. D. Wong, "LithoBench: Benchmarking AI computational lithography for semiconductor manufacturing," in *Proc. NeurIPS*, 2023.

[29] S. Zheng, Y. Ma, B. Yu, and M. D. Wong, "EMOGen: Enhancing mask optimization via pattern generation," in *Proc. DAC*, 2024.

[30] H. Yang and H. Ren, "Enabling scalable AI computational lithography with physics-inspired models," in *Proc. ASPDAC*, 2023, pp. 715–720.

[31] T. Ajayi, V. A. Chhabria, M. Fogaça, S. Hashemi, A. Hosny, A. B. Kahng, M. Kim, J. Lee, U. Mallappa, M. Neseem *et al.*, "Toward an open-source digital flow: First learnings from the openroad project," in *Proc. DAC*, 2019.