Ultrafast Source Mask Optimization via Conditional Discrete Diffusion

Guojin Chen, Zixiao Wang, Bei Yu, *Senior Member, IEEE*, David Z. Pan, *Fellow, IEEE*, and Martin D. F. Wong, *Fellow, IEEE*

***Abstract*—Source mask optimization (SMO) is vital for mitigating lithography imaging distortions caused by shrinking critical dimensions in integrated circuit fabrication. However, the computational intensity of SMO, involving multiple integrals in Abbe's theory, hinders its widespread adoption and advancement. In this article, we present Diff-SMO, a highly efficient and accurate SMO framework with a primary emphasis on enhancing source optimization techniques. Previous research was confined to mask optimization acceleration due to the constraints of the academia lithography model. Diff-SMO extends the scope of optimization by concurrently refining the intricate interplay between the source and mask. We first develop a GPU-accelerated lithography simulator grounded in Abbe's theory, enabling full GPU acceleration throughout the SMO process. Furthermore, we propose a discrete diffusion model for generating quasi-optimal sources, significantly improving computational efficiency. Our experimental results demonstrate exceptional imaging fidelity, surpassing the state-of-the-art, with over 200 times higher throughput compared to traditional SMO methods.**

***Index Terms*—Deep learning, design automation, design for manufacture.**

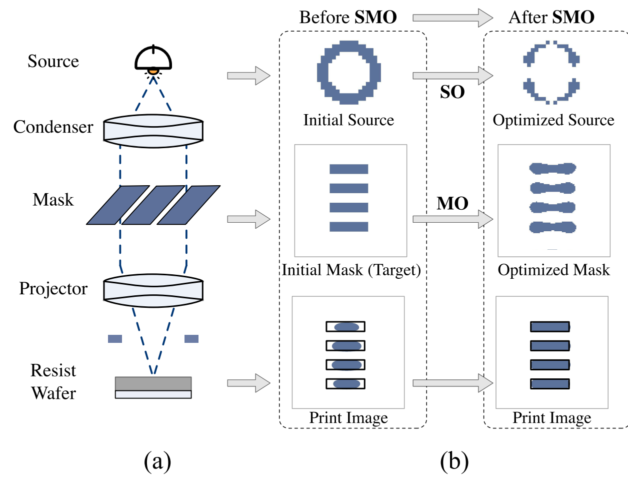

Fig. 1. (a) Illustration of DUV lithography system. (b) Flow of SMO. The pattern error of the printed image will be significantly reduced through SMO.

I. Introduction

Optical lithography plays a vital role in integrated circuit (IC) manufacturing, using light to selectively expose a photosensitive material and meticulously pattern IC's thin film layers. State-of-the-art (SOTA) resolution enhancement techniques like source mask optimization (SMO) and multiple patterning have facilitated the creation of complex ICs, characterized by smaller feature sizes and enhanced transistor densities. In essence, source optimization (SO) adjusts the incidence angles or intensity distributions of light rays emitted from the illumination system. Conversely, mask optimization (MO) regulates the amplitude of the transmitted light rays via mask, as shown in Fig. 1. Compared to MO, which is more commonly known as optical proximity correction (OPC), SMO can gain a larger manufacturing process window by adjusting the source intensity distribution. This feature has generated significant interest among semiconductor foundries, inciting extensive research to prolong the viability of 193-nm optical lithography [1], [2]. The advent of freeform illumination has enabled pixelated SMO techniques to enhance optimization freedom degrees, consequently boosting the imaging performance of lithography systems [3], [4]. However, the intricate source representation significantly affects the pixelated SMO method's performance. Additionally, the escalating density of IC layouts triggers a surge in the data volume processed by SMO algorithms. Pixel-based optimization introduces numerous optimization variables, thereby escalating the computational complexity of the optimization process. Collectively, these factors impose substantial challenges on the computational efficiency of SMO algorithms.

The advent of Hopkins' approach [5] and its simplified formulation, the "*sum of coherent systems*" (SOCS) [6], have triggered a significant reduction in MO computational complexity via singular value decomposition (SVD). Consequently, research has veered toward the exploration of distinct acceleration techniques for SO and MO [7], [8], [9], [10].

Recent years have seen a notable surge in research dedicated to MO acceleration [11], [12], [13], [14] using GPU and deep

Manuscript received 29 July 2023; revised 1 November 2023 and 8 January 2024; accepted 26 January 2024. Date of publication 2 February 2024; date of current version 20 June 2024. This work was supported in part by the Research Grants Council of Hong Kong, SAR, under Project CUHK14208021. This article was recommended by Associate Editor I. H.-R. Jiang. *(Corresponding author: Bei Yu.)*

Guojin Chen, Zixiao Wang, and Bei Yu are with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong (e-mail: byu@cse.cuhk.edu.hk).

David Z. Pan is with the Department of Electrical and Computer Engineering, University of Texas at Austin, Austin, TX 78712 USA.

Martin D. F. Wong is with the Department of Computer Science, Hong Kong Baptist University, Hong Kong.

Digital Object Identifier 10.1109/TCAD.2024.3361400

neural network (DNN) technologies, facilitated by lithography simulators grounded in the Hopkins model provided by the ICCAD 2013 contest [15]. GAN-OPC [11] utilizes generative adversarial networks (GANs) to predict initial solutions for traditional MO solvers [7]. DevelSet [12] employs a DNN to forecast the level set of masks, which are subsequently refined and optimized using a GPU-accelerated level set MO solver. A2ILT [13] employs on-neural-network inverse lithography techniques (ILT) and reinforcement learning to expedite MO. Multi-ILT [14] leverages multilevel ILT combined with sub-resolution assist features (SRAFs). These strategies, relying on DNNs to predict a superior initial solution followed by GPU-accelerated MO fine-tuning, yield impressive results in terms of both speed and MO outcome.

The illumination system of advanced lithography machines has made significant progress, achieving what is called freeform illumination. This intensity distribution of source map is composed of many pixelated illuminators, and the intensity of each pixel can be controlled by computer programs. The pixelated illumination system is necessary for SO. However, SO, which is dictated by Abbe's imaging approach, presents a challenge for the direct application of GPU acceleration techniques. Wang et al. [16] proposed to use compressive sensing to reduce the computational complexity of the SMO algorithm. ICC-CPU [17] puts forth a sampling-based imaging model employing heuristic algorithms to augment SMO efficiency and performance. SoulNet [18] utilizes an autoencoder to learn the model-based source correlating with each layout. Despite this, SoulNet falls short in accomplishing iterative SMO and offers limited optimization capabilities for the overall result. At present, while MO optimization time has been reduced to seconds for the ICCAD13 benchmark [15], CPU-based SO still requires several hours of optimization, creating a substantial bottleneck within the overall SMO workflow.

To navigate these challenges and foster advancements in SMO research, we introduce the *Diff-SMO* framework, composed of three key components. Initially, we have constructed a lithography algorithm tailored for GPU parallel acceleration, exploiting the intrinsic properties of the rigorous Abbe model. Subsequently, building on previous accomplishments in GPU/DNN acceleration for MO, we have achieved GPU acceleration of the SO algorithm (*GPU-SO*), culminating in full GPU acceleration of the entire SMO workflow. Lastly, we present *Diff-SO*, which incorporates a conditional discrete diffusion model for quick production of mask-aware, near-optimal sources. The diffusion process imitates the progressive turning off/on of the source in a discrete state space. The resultant solution of diffusion model provides an improved initial choice for GPU-SO to facilitate further precise optimization. By reducing the number of iterations required by GPU-SO, Diff-SO expedites the SMO process and enhances the optimization results. Our contributions can be summarized as follows.

1) We present the first academic effort, to our knowledge, to design a GPU-accelerated lithography simulator based on Abbe's aerial imaging theory.
2) We devise GPU-SO, a GPU acceleration algorithm specifically for SO, thus facilitating the comprehensive integration of GPUs across the entire SMO process.
3) We propose Diff-SO, an innovative conditional discrete diffusion model that generates near-optimal sources, enhancing the efficiency and performance of SMO significantly.
4) Our experimental data showcase the effectiveness of our method, demonstrating an average improvement of 9% over SOTA MO techniques. Furthermore, in comparison to prevalent CPU-based SOTA SMO methods, Diff-SMO provides an 8% performance increase along with a remarkable speed boost of nearly 225 times.

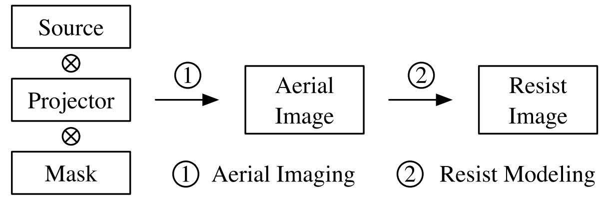

Fig. 2. Flowchart of lithography simulation, including aerial imaging and resist modeling.

II. Preliminaries

In this section, we will discuss the fundamental concepts of lithography imaging theory. We will specifically focus on Abbe's method, which serves as the foundation for SO algorithms. Additionally, we will explore Hopkins' method, a widely used approach for MO algorithms. Furthermore, we will define the SMO problem and delve into the evaluation metrics associated with it.

A. Lithography Simulation

In a typical lithography simulation, the lithographic imaging model comprises two essential components: 1) the illumination source and 2) the projector system, as depicted in Figs. 1 and 2. During the manufacturing process, the light rays propagate through the mask and produce diffracted light with feature pattern information. The intensity distribution of aerial image can be formulated via lithography theory [4] as

$$\begin{aligned} I(x,y) = &\iiiint\!\!\iint_{-\infty}^{\infty} J(f,g)\ \mathcal{F}(M)\big(f',g'\big)\mathcal{F}(M)^{*}\big(f'',g''\big) \\ &H\big(f+f',g+g'\big)H^{*}\big(f+f'',g+g''\big) \\ &\exp\big(-j2\pi((f'-f'')x+(g'-g'')y)\big) \\ &\mathrm{d}f\ \mathrm{d}g\ \mathrm{d}f'\ \mathrm{d}g'\ \mathrm{d}f''\ \mathrm{d}g'' \end{aligned} \tag{1}$$

where $\boldsymbol{I}$ is the aerial intensity distribution; $\boldsymbol{J}$ is the lithographic illumination source; $\boldsymbol{H}$ denotes the optical transfer function (OTF) of the projector system; $\mathcal{F}(\boldsymbol{M})$ represents the frequency spectrum of the mask pattern $\boldsymbol{M}$ obtained via $2-D$ fast Fourier transform (`FFT`) $\mathcal{F}(\cdot)$; * is the Hermitian transpose operator; (x,y) denotes the spatial coordinates on the image plane; (f,g), (f',g') and (f'',g'') represent the normalized-frequency-domain coordinates of the pupil, mask spectrum and its conjugate, respectively. Two formulas are used to calculate the partially coherent aerial image formation in (1).

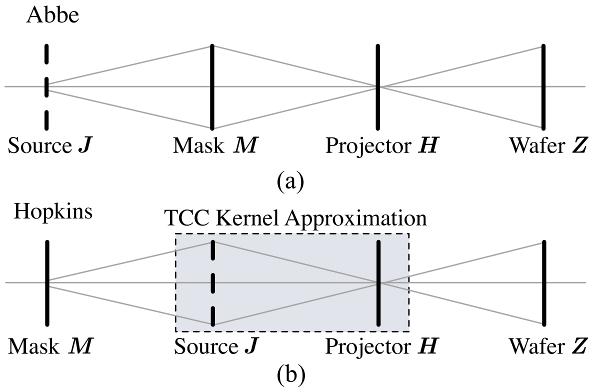

Fig. 3. Comparison of Abbe's (a) and Hopkins' (b) imaging. The source is discretized and the contribution from each source point toward the final aerial image is summed in the Abbe's formulation. In the Hopkins' approach, the transmission cross correlation matrix (TCC) of the source with the projector and its complex conjugate is computed for an arbitrary mask geometry with a fixed size.

As illustrated in Fig. 3, one is Abbe's approach [19] and the other is Hopkins' method [5].

Abbe's Approach: In Abbe's approach, the source illuminator is discretized, as illustrated in Fig. 3(a). The contribution from each source point toward the final aerial image is calculated independently and then summed. For convenience, we introduce the *illumination cross-coefficients* (***ICC***) as

$$ICC(x, y; f, g) = \Big| \iint_{-\infty}^{\infty} H\big(f+f', g+g'\big)\mathcal{F}(M)\big(f', g'\big) \exp\big(-j2\pi(f'x+g'y)\big)\, \mathrm{d}f'\, \mathrm{d}g'\Big|^2. \quad (2)$$

Equation (1) can be formulated in Abbe's approach

$$I(x, y) = \iint_{-\infty}^{\infty} J(f, g)\, ICC(x, y; f, g)\, \mathrm{d}f\, \mathrm{d}g. \quad (3)$$

Hopkins' Approach: As depicted in Fig. 3(b), Hopkins' approach separating the illuminator and projector system from mask has the following expression that is:

$$I(x, y) = \iiiint_{-\infty}^{\infty} \mathcal{T}\big(f', g'; f'', g''\big)\mathcal{F}(M)\big(f', g'\big)\mathcal{F}(M)^*\big(f'', g''\big) \exp\big(-j2\pi((f'-f'')x + (g'-g'')y)\big)\mathrm{d}f'\mathrm{d}g'\mathrm{d}f''\mathrm{d}g'' \quad (4)$$

where $\mathcal{T}$ is the *transmission cross-coefficients* ***TCC*** given by

$$\mathrm{TCC}\big(f', g'; f'', g''\big) = \iint_{-\infty}^{\infty} J(f, g)\, H\big(f+f', g+g'\big)H^*\big(f+f'', g+g''\big)\mathrm{d}f\mathrm{d}g. \quad (5)$$

To reduce the calculation complexity, an approximation solution for the Hopkins imaging equations called *Sum of coherent source* (SOCS) decomposes the ***TCC*** spectrum by applying SVD as follows:

$$\mathrm{TCC}\big(f', g'; f'', g''\big) = \sum_{q=1}^{\infty} \kappa_q \Phi_q\big(f', g'\big)\Phi_q^*\big(f'', g''\big) \quad (6)$$

where κ_q and $\boldsymbol{\Phi_q}$ are q'th eigenvalue and eigenvector of ***TCC***. Because the eigenvalue κ_q is rapidly decay with increasing q, we can keep the Q largest eigenvalues for faster calculation. Substituting (6) into (4) then performing Inverse Fast Fourier Transform (IFFT), the SOCS can be expressed to spatial position via (7) that is

$$I(x, y) = \sum_{q=1}^{Q} \kappa_q \big|\boldsymbol{\phi}_q(x, y) \ \otimes M(x, y)\big|^2 \quad (7)$$

where $\boldsymbol{\phi}_q(x, y)$ and $M(x, y)$ are the spatial distribution of $\boldsymbol{\Phi_q}$ and $\mathcal{F}(\boldsymbol{M})$, respectively. $\otimes$ and $|\cdot|$ are the convolution and absolute operator.

Hopkins' Approach Versus Abbe's Approach: Hopkins' method (7) applies the SOCS to decrease computational complexity from $\mathcal{O}(n^6)$ to $\mathcal{O}(Q \times n^4)$, considering $M \in \mathbb{R}^{n\times n}$. Importantly, it separates mask from optical system, rendering Hopkins' method more suitable for simulating images while optimizing masks. With constant optical imaging conditions, Hopkins' method outperforms Abbe's in runtime, making it a favored choice in MO algorithms [7], [11], [12], [13], [14]. In contrast, Abbe's method in (3), sums up the impact of all source points to generate the final aerial image. This method is inherently suited for SO due to the source's discretization. Consequently, to achieve SO and SMO, devising a lithography simulator founded on Abbe's method is a prerequisite. The comparison of Hopkins' and Abbe's imaging is illustrated in Fig. 3.

B. Evaluation Metrics

Definition 1 (Squared L_2 Error): Given target pattern $\boldsymbol{Z}_t$ and wafer pattern under nominal process condition $\boldsymbol{Z}$, the squared L_2 error is calculated as $\|\boldsymbol{Z} - \boldsymbol{Z}_t\|_2^2$.

Definition 2 (Process Variation Band (PVB)): PVB [15] is used in manufacturing to represent the expected range of variation in a production process. PVB denotes the XOR area between the aerial images $\boldsymbol{Z}_{in}$ and $\boldsymbol{Z}_{out}$ under the *min* and *max* lithography process conditions

Definition 3 (Edge Placement Error (EPE)): EPE [15] refers to the deviation between the intended position of a feature on a wafer and its actual position after lithography.

C. Problem Formulation

To formulate the problem of SMO, we first introduce the necessary terminologies. Given a target image represented as $\boldsymbol{Z}_t$, we define a function F which represents the weighted sum of three components: 1) the L_2 error; 2) PVB; and 3) EPE that arise after the lithography process. Under given circumstances: 1) if the source parameters J and the projector system are predetermined, the task of MO is to discern the ideal mask configuration that minimizes F and 2) conversely, with a fixed mask in place, SO focuses on adjusting the source J to achieve a minimal F. The overarching goal of SMO is to concurrently identify the optimal configurations for both the source, denoted as $\hat{\boldsymbol{J}}$, and the mask, denoted as $\hat{\boldsymbol{M}}$, to minimize F. This can be mathematically represented as

$$\big(\hat{\boldsymbol{J}}, \hat{\boldsymbol{M}}\big) = \arg\min_{(\boldsymbol{J},\boldsymbol{M})} F(\boldsymbol{J}, \boldsymbol{M}). \quad (8)$$

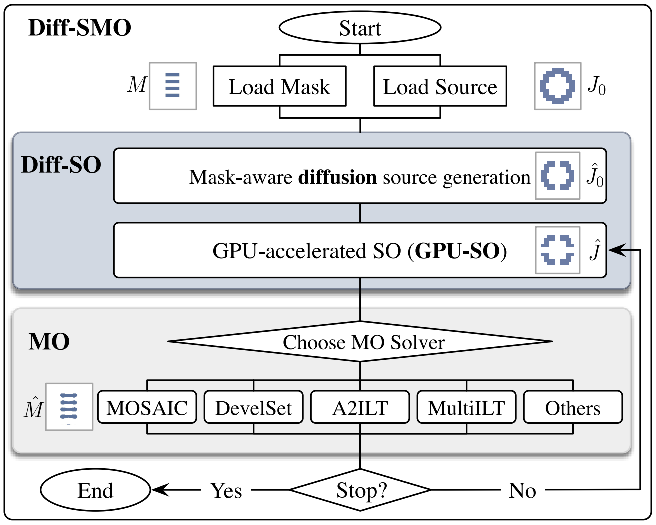

Fig. 4. Flowchart of the Diff-SMO framework.

Algorithm 1 GPU-Accelerated Abbe Aerial Imaging

Input: Source pattern $\boldsymbol{J}$; Mask pattern $\boldsymbol{M}$.
Output: Aerial image $\boldsymbol{I}$.

1: **function** ABBE_LITHO($\boldsymbol{J}$, $\boldsymbol{M}$)
2: $\quad\boldsymbol{J} \leftarrow$ Select $\boldsymbol{J}$ where $\boldsymbol{J} > 10^{-5}$;
3: $\quad\mathcal{F}(\boldsymbol{M}) \leftarrow \texttt{fftshift}(\texttt{fft2}(\boldsymbol{M}))$;
4: $\quad J_{\text{sum}} \leftarrow \sum \boldsymbol{J}$;
5: $\quad$**for all** source points $s_i \in \boldsymbol{J}$, parallelly **do**
6: $\qquad s_i \leftarrow$ Assign a GPU thread to s_i;
7: $\qquad \boldsymbol{H}_s \leftarrow$ Transfer function for s_i ▷ Eq. (12);
8: $\qquad \mathcal{F}(\boldsymbol{M})_s \leftarrow$ Valid spectrum for s_i; ▷ Eq. (13)
9: $\qquad \boldsymbol{E}_s \leftarrow \mathcal{F}^{-1}(\boldsymbol{H}_s \odot \mathcal{F}(\boldsymbol{M})_s)$;
10: $\qquad \boldsymbol{I}_i \leftarrow s_i \cdot |\boldsymbol{E}_s \odot \boldsymbol{E}_s^*|$;
11: $\quad$**end for**
12: $\quad\boldsymbol{I} \leftarrow \sum \boldsymbol{I}_i / J_{\text{sum}}$; ▷ Normalization
13: $\quad$**return** $\boldsymbol{I}$;
14: **end function**

Algorithm 2 GPU-Accelerated SO

Input: Initial source $\boldsymbol{J}_0$; Mask $\boldsymbol{M}$; Target $\boldsymbol{Z}_t$.
Output: Optimal source $\hat{\boldsymbol{J}}$.

$\boldsymbol{J}_\theta \leftarrow$ Initialize $\boldsymbol{J}_\theta$ using Eq. (14);
repeat
$\quad\boldsymbol{J} \leftarrow \texttt{Sigmoid}(\alpha \cdot \boldsymbol{J}_\theta)$; ▷ Eq. (15)
$\quad\boldsymbol{I} \leftarrow \texttt{Abbe\_litho}(\boldsymbol{J}, \boldsymbol{M})$; ▷ Algorithm 1
$\quad\boldsymbol{Z} \leftarrow \texttt{Sigmoid}(\beta \cdot (\boldsymbol{I} - I_{tr}))$;
$\quad L \leftarrow \|\boldsymbol{Z} - \boldsymbol{Z}_t\|^2$;
$\quad\boldsymbol{J}_\theta \leftarrow \boldsymbol{J}_\theta - lr \cdot \texttt{Adam}(\partial L / \partial \boldsymbol{J}_\theta)$;
until $\Delta L < \delta$
return Optimal source $\hat{\boldsymbol{J}}$;

III. Algorithm

This section outlines the Diff-SMO framework, as visualized in Fig. 4. Initially, we address the GPU-accelerated SO algorithm in Section III-A. Then, Section III-B introduces the diffusion model, crucial for mask-aware source creation. Finally, Section III-C delves into the entire Diff-SMO process. We simplify our terminology as follows: *GPU-SO* for the GPU-accelerated SO algorithm, *Diff-SO* for the application of diffusion in source generation and its subsequent optimization with GPU-SO, and *Diff-SMO* for the combined framework of Diff-SO and the MO algorithm, as depicted in Fig. 4.

A. GPU-Accelerated SO Algorithm

Computation Complexity Analysis of Vanilla SO: In numerical terms, (3) must be resolved for a discrete number of mutually independent source points as follows:

$$I(x_i, y_i) = \sum_{f=1}^{N_s} \sum_{g=1}^{N_s} J(f, g)\ ICC(x_i, y_i; f, g) \tag{9}$$

where N_s denotes the lateral dimension's pixelated source sampling number. We designate the S operator as the reshape operation

$$\mathbb{R}^{n\times n} \xrightarrow{\text{S}} \mathbb{R}^{n^2\times 1} \xrightarrow{\text{S}^{-1}} \mathbb{R}^{n\times n}. \tag{10}$$

As a result, (9) can be translated into (11) via matrix multiplication

$$\boldsymbol{I} = \text{S}^{-1}(\boldsymbol{ICC}\, \text{S}(\boldsymbol{J})) \tag{11}$$

where $\boldsymbol{I} \in \mathbb{R}^{N\times N}$, $\boldsymbol{ICC} \in \mathbb{R}^{N^2\times N_s^2}$, and $\boldsymbol{J} \in \mathbb{R}^{N_s\times N_s}$, respectively, and N is the mask spatial domain's sampling number. The computational complexity is $\mathcal{O}(n^6)$. In our implementation, specifically, $N = 2048$ and N_s ranges from 35 to 58. The computation of large matrix products, between $\boldsymbol{ICC}$ and $\boldsymbol{J}$, incurs significant time cost and necessitates careful memory allocation. This computational demand serves as a substantial impediment to the execution time of the forward aerial imaging process. Furthermore, the situation worsens during gradient calculation, where the process often necessitates a duration several magnitudes greater than the forward computation time.

GPU-Accelerated Abbe Aerial Imaging: Historically, the literature has largely depended on acceleration techniques employing compressive sensing or heuristic approaches [17] to selectively observe certain points from the extensive $\boldsymbol{ICC}$ matrix. These strategies often tradeoff a degree of accuracy for computation speed enhancement. However, our approach distinguishes itself by leveraging GPU acceleration to fundamentally address the slow computation issue inherent in the Abbe model. Importantly, we recognize that each discrete light source computation in the Abbe model, as per (9), is independent and noninterfering. This characteristic renders the Abbe model particularly amenable to parallel computing acceleration via GPUs.

The algorithm outlined in Algorithm 1 presents a new approach to GPU-accelerated aerial imaging generation based on Abbe's approach. It starts with noise elimination and the selective retention of source points that contribute to the generation process, as indicated in line 2. By applying a threshold, we can reduce computational overhead caused by zero-valued sources. The method contrasts with prior techniques which relied on fixed-dimensional matrix multiplication. Instead, our algorithm allows for finer acceleration granularity at the

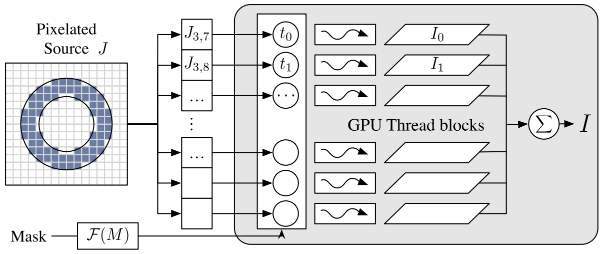

Fig. 5. Illustration of GPU-accelerated Abbe's aerial imaging in Algorithm 1. Each discretized source is assigned to an individual thread for computation. The contributions of all source points are summed up to generate the aerial image.

individual source level, providing both increased acceleration potential and flexibility. This is achieved by allowing independent filtering and computation for each source point, as detailed in line 5 and Fig. 5. Building on Abbe's imaging theory, the projector transfer function is computed as per line 7, and can be visualized as a low-pass filter of the spatial frequency domain, given by (12)

$$H(f, g) = \begin{cases} 1, & \sqrt{f^2 + g^2} \leq NA/\lambda \\ 0, & \text{otherwise} \end{cases}. \tag{12}$$

Similarly, $J(f, g)$ is expressed by (13), where NA/λ is the cut-off spatial frequency defined by the wavelength λ and numerical aperture NA of the projection system

$$J(f, g) = \begin{cases} J(f, g), & \sqrt{f^2 + g^2} \leq \sigma NA/\lambda \\ 0, & \text{otherwise} \end{cases}. \tag{13}$$

Further, the partial coherence factor σ is determined by the size ratio of the source image and the pupil. $\sigma NA/\lambda$ represents the maximum spatial working region of the source, relative to the spatial frequency space of the image domain, as evaluated in line 8. By applying the `IFFT` to the result of an element-wise product ($\odot$) of $\boldsymbol{H}_s$ and $\mathcal{F}(\boldsymbol{M})_s$ as shown in line 9, we can compute the contribution from each point. The final aerial image $\boldsymbol{I}$ is obtained by summing up these individual contributions from each source point, each calculated within a separate thread, as summarized in line 12.

GPU-Accelerated SO: The GPU-accelerated SO algorithm, shown in Algorithm 2, employs freeform illumination in the pixelated SMO process, which grids the source pattern $\boldsymbol{J}$ into $N_s \times N_s$ pixels, as is depicted in Fig. 5. Each pixel's value signifies the intensity of the associated source point, falling within the range of [0, 1]. To ensure the SMO flow's differentiability, we introduce the source parameters $\boldsymbol{J}_\theta$, as initialized by

$$J_\theta(f, g) = \begin{cases} 1, & J_0(f, g) = 1 \\ -1, & J_0(f, g) = 0 \end{cases} \tag{14}$$

where $\boldsymbol{J}_\theta \in \mathbb{R}^{N_s \times N_s}$ and can hold any real value in the range $(-\infty, \infty)$. Meanwhile, $\boldsymbol{J}_0$ represents the initial pixelated source pattern derived from the parametric representations of source templates. Some examples of $\boldsymbol{J}_0$ are visualized in Fig. 6. During the optimization process, source $\boldsymbol{J}$ is obtained by applying the `Sigmoid` function to the parameters $\boldsymbol{J}_\theta$

$$\boldsymbol{J} = \text{Sigmoid}(\alpha \cdot \boldsymbol{J}_\theta) = \frac{1}{1 + \exp(-\alpha \cdot \boldsymbol{J}_\theta)} \tag{15}$$

where α is a hyperparameter to define the steepness of the `Sigmoid` function. Afterward, the aerial image $\boldsymbol{I}$ is calculated using the `Abbe_litho` function (introduced in Algorithm 1), with $\boldsymbol{J}$ and mask $\boldsymbol{M}$ as inputs. The Algorithm 2 utilizes a simple resist model in line 5, which applies the `Sigmoid` function to $\boldsymbol{I}$

$$\boldsymbol{Z} = \frac{1}{1 + \exp(-\beta \cdot (\boldsymbol{I} - I_{tr}))} \tag{16}$$

where $\boldsymbol{Z}$ is the resist pattern; I_{tr} is the intensity threshold; β is a hyperparameter for steepness. In line 6, the mean-squared error loss L is then computed between the predicted resist image $\boldsymbol{Z}$ and the target $\boldsymbol{Z}_t$, where $L = \|\boldsymbol{Z} - \boldsymbol{Z}_t\|^2$. The source parameters $\boldsymbol{J}_\theta$ are subsequently updated in line 7 via gradient backpropagation using the `Adam` optimizer. The optimization process persists until the change in loss ΔL between successive iterations falls below a threshold δ.

GPU-Accelerated SO Versus Multicore CPU-Accelerated SO: Central processing units (CPUs) act as the primary computational elements, executing essential logical, arithmetic, and input/output operations. They also manage the distribution of tasks to different components and subsystems in a computer system. In modern times, CPUs are frequently designed with a multicore structure, which means multiple processing units are housed within a single IC. This design approach not only conserves power but also improves performance, enabling efficient parallel processing of simultaneous tasks.

The previously mentioned GPU-accelerated SO's parallel algorithm can be fully migrated to a multicore CPU. We have also implemented a CPU parallel acceleration algorithm based on multiprocessing. However, compared to the multicore CPU-accelerated SO, the GPU-accelerated SO still has the following advantages. First, because the entire SO optimization process doesn't require extensive IO reading operations, it can utilize GPU pinned memory to speed up computations. Additionally, GPUs offer greater memory bandwidth, saving significant time during memory access. Furthermore, for large kernel size operations, GPUs can compute FFT/IFFT more quickly, which is frequently invoked throughout the SO process. Also, previous SOTA MO researches has shown that GPU-accelerated MO is effective. A comprehensive GPU-accelerated SMO algorithm can greatly reduce the time wasted transferring data between the CPU and GPU. Lastly, multicore CPUs might be more expensive than GPUs. For tasks like SO that require massive parallelism, GPUs are not only more suitable but also more cost-effective than CPUs.

B. Diffusion Models for Mask-Aware Source Generation

Currently, the estimation of our GPU-accelerated SO is initiated from a template source as defined in (14) and illustrated in Fig. 6. Intuitively, a mask-aware initialization can reduce the number of iterations and accelerate the optimization process. The task of exact source initialization for our GPU-accelerated SO is difficult to formulate using traditional rule-based methodology. However, an approximate initialization is sufficient in this case. Therefore, we resort to learning-based methods to solve this problem.

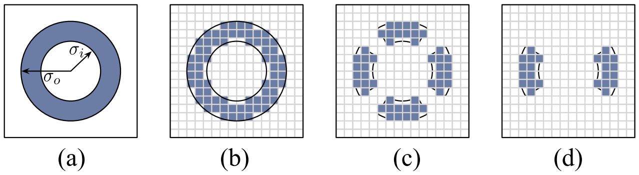

Fig. 6. Different representations and types of source templates. (a) Parametric representation of annular illuminator source, defined by outer and inner radii σ_o and σ_i; (b) pixelated representation of discrete annular source; (c) Quasar source; and (d) dipole source.

The source initialization can be formulated as a conditional binary discriminative task: given the mask and an empty source matrix, the model learns to determine which pixels in the source matrix should be turned on/off. Using a discriminative model to judge the state of every source pixel can be a natural choice but the large amount of discriminative heads can be too heavy. Instead of that, we introduce an efficient generative model, namely, the conditional discrete diffusion model, in our method to generate source matrix from given condition.

Conditional Discrete Diffusion Denoising Model: Typical generative models, such as GANs and diffusion model [20], are commonly applied in the continuous value space, where each generated pixel is represented as a floating-point number within a predefined range. However, in our source initialization case, the state of each pixel in the source matrix is binary (on/off). Applying a threshold to a continuous pixel value and converting it into a binary value may result in information truncation and compromise performance. In contrast to the traditional diffusion models used in computer vision, we have made several important customizations to enhance the diffusion model and directly synthesize a discrete source matrix from the given mask. To derive these enhancements, we first reformulate the problem.

In our approach, we employ a forward process to simulate the gradual turning off of the light in the given source matrix. Similarly, we utilize a reverse process to mimic the gradual turning on process. Our conditional discrete diffusion model defines a Markov chain to represent both the forward turn-off process and the reverse turn-on process, as illustrated in Fig. 7. The lengths of both processes are denoted as K. During the kth step of the forward process, $x_k \in \{0, 1\}$ represents an entry in the source matrix $\boldsymbol{J}$. The forward step can be described by the equation

$$\boldsymbol{q}(\boldsymbol{x}_k \mid \boldsymbol{x}_{k-1}) := \mathrm{Cat}\big(\boldsymbol{x}_k; \boldsymbol{p} = \boldsymbol{x}_{k-1}\boldsymbol{Q}_k\big) \tag{17}$$

where $\boldsymbol{x}_k$ is the one-hot version of the entry x_k, $\mathrm{Cat}(\boldsymbol{x}; \boldsymbol{p})$ is a categorical distribution over the row vector $\boldsymbol{x}$ with probabilities given by the row vector $\boldsymbol{p}$, $\boldsymbol{x}_{k-1}\boldsymbol{Q}_k$ can be understood as a row vector-matrix product and the transition probability matrix $[\boldsymbol{Q}_k]_{ij} = q(x_k = j | x_{k-1} = i)$ is defined to describe the state transition probability for each $\boldsymbol{x}$ at the kth step. The $\boldsymbol{Q}_k$ is applied to each entry in the source matrix independently and $\boldsymbol{q}$ factorizes over these higher dimensions as well. On the other side, in the reverse process, given the mask $\boldsymbol{M}$, the neural network aims to predict the conditional categorical distribution probability $\boldsymbol{p}_\theta(\boldsymbol{x}_{k-1}|\boldsymbol{x}_k, \boldsymbol{M})$ over each pixel to recover the original source matrix.

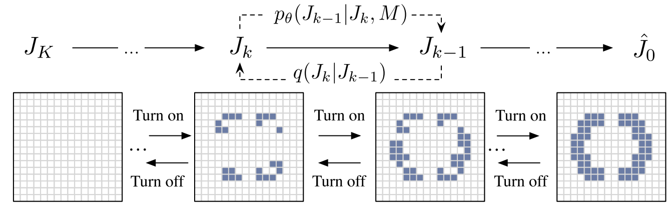

Fig. 7. Illustration of forward turn off process and reverse turn on process. $\boldsymbol{J}_K$ is an all-zero source matrix and $\hat{\boldsymbol{J}}_0$ is the mask-aware source matrix, which will be used as the initialization in our GPU-accelerated SO algorithm.

Algorithm 3 Training of Discrete Diffusion Model

Input: Ground Truth $\hat{\boldsymbol{J}}$; Source template $\boldsymbol{J}_0$; Mask $\boldsymbol{M}$.
Output: Well-trained model p_θ.

repeat
  $k \leftarrow \texttt{Random}(1, K)$;
  $\boldsymbol{J}_k \leftarrow \texttt{Turn\_off}(\hat{\boldsymbol{J}}, k)$; ▷ Turn_off, Eq. (22)
  $\hat{\boldsymbol{J}}_0 \leftarrow p_\theta(\boldsymbol{J}_k, \boldsymbol{J}_0, \boldsymbol{M}, k)$;
  $L \leftarrow \texttt{Loss}(\hat{\boldsymbol{J}}_0, \hat{\boldsymbol{J}}, \boldsymbol{J}_k)$; ▷ Eq. (21)
  $p_\theta \leftarrow \texttt{Update}(p_\theta, L)$;
until Finished
return Well-trained model p_θ;

The choice of transition probability matrix $\boldsymbol{Q}_k$ is critical. As we described before, in the forward turn-off process, the stationary state of every entry in the source matrix should be off (0 in our case) when k becomes large. So given any $\boldsymbol{x}_0$ and mask $\boldsymbol{M}$, the distribution of every entry $\boldsymbol{x}_k$ should follows:

$$\boldsymbol{q}(\boldsymbol{x}_k|\boldsymbol{x}_0, \boldsymbol{M}) \rightarrow [1, 0], \text{ when } k \rightarrow K. \tag{18}$$

Therefore, we design a transition matrix $\boldsymbol{Q}_k$ with an absorbing state 0 for the turn off process

$$\boldsymbol{Q}_k = \begin{bmatrix} 1 & 0 \\ \beta_k & 1 - \beta_k \end{bmatrix} \tag{19}$$

where $\beta_k \in (0, 1)$ is the hyperparameter controlling the flipping probability. In order to ensure that at the k of K step, (k/K) information about given source matrix is lost, following the suggestion of [21], we use an increasing noise schedule for β_k:

$$\beta_k = \frac{1}{K - k + 1}, \quad k = 1, \ldots, K. \tag{20}$$

Training Diffusion Model: To training the conditional discrete diffusion model for source matrix generation, the training objective at step k is to minimize the loss function

$$L = D_{\mathrm{KL}}\big(\boldsymbol{q}(\boldsymbol{x}_{k-1}|\boldsymbol{x}_k, \boldsymbol{x}_0) \| \boldsymbol{p}_\theta(\boldsymbol{x}_{k-1}|\boldsymbol{x}_k, \boldsymbol{M})\big) - \eta \log \boldsymbol{p}_{\boldsymbol{\theta}}(\boldsymbol{x}_0|\boldsymbol{x}_k, \boldsymbol{M}) \tag{21}$$

where η is a hyperparameter to balance the loss terms.

Given a source matrix $\boldsymbol{J}_0$, we randomly sample a target step k from an uniform distribution that is defined from 1 to K first, and expect to get a noisy sample $\boldsymbol{J}_k$. Fortunately, we can explicitly derive that $\boldsymbol{x}_k$ obeys the following categorical distribution:

$$\boldsymbol{q}(\boldsymbol{x}_k|\boldsymbol{x}_0) = \mathrm{Cat}\Big(\boldsymbol{x}_k; \boldsymbol{p} = \boldsymbol{x}_0\overline{\boldsymbol{Q}}_k\Big) \tag{22}$$

Algorithm 4 Generating Mask-Aware Source

Input: Source template $\boldsymbol{J}_0$; Mask $\boldsymbol{M}$; Model p_θ.
Output: Estimated mask-aware source $\hat{\boldsymbol{J}}_0$.

$\boldsymbol{J}_k \leftarrow$ `All_zero_matrix`;
for $k \in \{K, \ldots, 1\}$ **do**
  $\hat{\boldsymbol{J}}_0 \leftarrow p_\theta(\boldsymbol{J}_k, \boldsymbol{J}_0, \boldsymbol{M}, k)$;
  $\boldsymbol{J}_k \leftarrow \boldsymbol{J}_{k-1}(\hat{\boldsymbol{J}}_0, \boldsymbol{J}_k)$; ▷ `Turn_on`, Eq. (23)
end for
return Estimated mask-aware source $\hat{\boldsymbol{J}}_0$;

where $\overline{\boldsymbol{Q}}_k = \boldsymbol{Q}_1\boldsymbol{Q}_2 \ldots \boldsymbol{Q}_k$. Then, we can directly sample from the above distribution to obtain $\boldsymbol{J}_k$, instead of turning off k times.

After sampling $\boldsymbol{J}_k$, we extract the real $\widehat{\boldsymbol{M}}$ and imaginary $\widetilde{\boldsymbol{M}}$ parts from the spectrum of given mask $\boldsymbol{M}$. Then, we concatenate $\boldsymbol{J}_k$, $\widehat{\boldsymbol{M}}$, $\widetilde{\boldsymbol{M}}$ and the template source together and feed them into the neural network with the embedding of the time step k. The template source is defined in (14). The neural network will predict the logits of the posterior distribution $\boldsymbol{p_\theta}(\boldsymbol{x}_0|\boldsymbol{x}_k, \boldsymbol{M})$, and then $\boldsymbol{p_\theta}(\boldsymbol{x}_{k-1}|\boldsymbol{x}_k, \boldsymbol{M})$ can be calculated as following:

$$\boldsymbol{p_\theta}(\boldsymbol{x}_{k-1}|\boldsymbol{x}_k, \boldsymbol{M}) = \sum_{\widetilde{\boldsymbol{x}}_0} \boldsymbol{q}(\boldsymbol{x}_{k-1}|\boldsymbol{x}_k, \widetilde{\boldsymbol{x}}_0)\boldsymbol{p_\theta}(\widetilde{\boldsymbol{x}}_0|\boldsymbol{x}_k, \boldsymbol{M}) \tag{23}$$

where the term $\widetilde{\boldsymbol{x}}_0$ will visit every possible state of $\boldsymbol{x}_0$. And $\boldsymbol{q}(\boldsymbol{x}_{k-1}|\boldsymbol{x}_k, \widetilde{\boldsymbol{x}}_0)$ has a closed form according to (17) and Bayes' theorem

$$\boldsymbol{q}(\boldsymbol{x}_{k-1}|\boldsymbol{x}_k, \boldsymbol{x}_0) = \mathrm{Cat}\left(\boldsymbol{x}_{k-1}; \boldsymbol{p} = \frac{\boldsymbol{x}_k\boldsymbol{Q}_k^\top \odot \boldsymbol{x}_0\overline{\boldsymbol{Q}}_{k-1}}{\boldsymbol{x}_0\overline{\boldsymbol{Q}}_k\boldsymbol{x}_k^\top}\right) \tag{24}$$

where $\odot$ is a pixel-wise multiplication.

So far, all items in the loss function have been obtained, and the neural network can be trained by the commonly used gradient descent method. The training process is summarized in Algorithm 3.

Generating Mask-Aware Source Matrix: Once the model is well-trained, given the mask $\boldsymbol{M}$ and the type of template source, we can extract an mask-aware source matrix by starting from a all-zero (dark) source matrix and gradually turn on the pixel with the reverse procedure. The generating process can be expressed by

$$p_\theta\left(\hat{\boldsymbol{J}}_0|\boldsymbol{J}_K, \boldsymbol{M}\right) = p_\theta\left(\hat{\boldsymbol{J}}_0|\boldsymbol{J}_1, \boldsymbol{M}\right)\prod_{k=2}^{K} p_\theta(\boldsymbol{J}_{k-1}|\boldsymbol{J}_k, \boldsymbol{M}) \tag{25}$$

where $\boldsymbol{J}_k$ is the estimated source matrix at step k and $\hat{\boldsymbol{J}}_0$ is the newly sampled source matrix. When generation finished, the generated source matrix $\hat{\boldsymbol{J}}_0$ is naturally a binary one where each entry equals either zero or one. And the generated source matrix will be utilized as the initialization of our GPU-accelerated SO. The inference process is summarized in Algorithm 4.

C. Diff-SMO Flow

As illustrated in Fig. 4, the process commences with the loading of the source $\boldsymbol{J}_0$ and mask $\boldsymbol{M}$. By using the source template and mask spectrum as conditional factors, a well-trained diffusion model is deployed to efficiently generate a near-optimal source approximation, $\hat{\boldsymbol{J}}_0$. This approximation is then fed into our GPU-accelerated SO solver for a more precise optimization, leading to the optimal source $\hat{\boldsymbol{J}}$. When the predetermined stopping criteria for SO are satisfied, a $\boldsymbol{TCC}$ matrix will be calculated by integrating the optimized source with the pupil data following (5). Subsequently, the $\boldsymbol{TCC}$ matrix is utilized in MO solvers, as referenced in [7], [12], [13] and [14], to further optimize the mask. The refined mask and source are iteratively fed back into the Diff-SMO optimizer, guiding the optimization process until convergence is achieved for both the source and mask components.

IV. Experiments

The Diff-SMO framework, developed in the `Pytorch` framework, was tested on a Linux System using a single Nvidia RTX 3090 GPU card. The multicore CPU-accelerated SO algorithm runs on AMD EPYC 9554P with 64 cores using the 7-nm Zen architecture. This architecture supports up to 64 cores and 128 threads. Hyperthreading and Mult*i*threading technologies allow each core to have two virtual cores, enhancing performance. We conducted a calibration of the lithographic system to match the simulator [15] used in ICCAD 2013 before starting our experiments. The hyperparameter settings are as follows: $\sigma = 1$; $\alpha = 8$; $\beta = 30$; $I_{tr} = 0.225$; $\delta = 10^{-4}$; $\lambda = 193$ nm; $NA = 1.35$; $\eta = 0.001$; $K = 100$; inner radius $\sigma_i = 0.63$; outer radius $\sigma_o = 0.95$; and refractive index is 1.414. For simplicity, we denote squared L2 error, PVB, and EPE as "L2," "PVB," and "EPE," respectively.

We evaluate the performance of the Diff-SMO framework on the ICCAD 2013 CAD contest benchmark [15] using ten $4-\mu\mathrm{m}^2$ tiles of 32-nm industrial layouts. The training set for the diffusion model is obtained from [11], which contains 4K $4-\mu\mathrm{m}^2$ generated tiles following the same design rules as in [15]. The optimal ground-truth source for each mask is obtained by fully optimizing the template source with GPU-SO, using the corresponding mask as input.

Following [20], [22], the U-Net is selected as our backbone for diffusion model. The model predicts the posterior distribution during the reverse diffusion process. All input images are resized to 32×32 and consists of four feature map resolutions: $[32 \times 32, 16 \times 16, 8 \times 8, 4 \times 4]$. Each resolution level contains two convolutional residual blocks, with the number of convolution channels being [128, 256, 256, 256] correspondingly. A self-attention block is placed at the 16×16 resolution level. Additionally, the time step k is incorporated into each residual block through the sinusoidal position embedding [23].

A. Result Comparison With the SOTA

In this section, we will first validate the efficiency of Diff-SO. Following that, we will compare Diff-SO with the previous SOTA SO methods. The approach is to use different MO methods as a foundation, combined with various SO methods, to validate the performance of different SMO methods. Then, we compared the EPE and Runtime performance of different SMO combinations. Finally, we compared the

TABLE I
EFFECTIVENESS OF DIFF-SO

Bench. / MO algo.		MOSAIC[DAC14]				DevelSet [ICCAD21]				A2ILT [DAC22]				Multi-ILT [DAC23]			
ID	Area	w/o SO [7]		w/ Diff-SO		w/o SO [12]		w/ Diff-SO		w/o SO [13]		w/ Diff-SO		w/o SO [14]		w/ Diff-SO	
		L2	PVB	L2	PVB	L2	PVB	L2	PVB	L2	PVB	L2	PVB	L2	PVB	L2	PVB
case1	215344	49893	65534	43788	57186	49142	59607	42793	54584	45284	59940	42150	53993	42807	50759	39164	46183
case2	169280	50369	48230	39993	48918	34489	52012	32819	47697	34044	51988	31048	47021	35924	44255	28474	38287
case3	213504	81007	108608	78594	83711	93498	76558	76623	72336	92505	91261	74764	84465	68246	72447	65908	68036
case4	82560	20044	28285	18951	27186	18682	29047	16934	24618	21644	29017	18876	26265	15125	23620	15288	23468
case5	281958	44656	58835	38974	57310	44256	58085	35697	54396	38082	61601	34098	57694	30288	50933	29015	49443
case6	286234	57375	48739	40312	48991	41730	53410	37476	49325	42068	53620	36976	48776	30911	45811	29871	44498
case7	229149	37221	43490	24986	45904	25797	46606	21676	45213	21947	49053	20023	45964	23711	46938	15692	38149
case8	128544	19782	22846	15879	22997	15460	24836	15088	22597	15668	23853	14134	21987	14576	22314	13872	21234
case9	317581	55399	66331	47093	62007	50834	64950	45685	61327	46973	68442	43983	61808	38220	59620	36237	54982
case10	102400	24381	18097	12172	19651	10140	21619	11073	19874	10450	19950	9040	19365	12473	19394	8943	17996
Avg.	202655	44013	50900	36074	47386	38403	48673	33586	45197	36867	50873	32509	46734	31228	43609	**28246**	**40228**
Ratio	-	1	1	0.820	0.931	1	1	0.875	0.929	1	1	0.882	0.919	1	1	0.905	0.922

L2 and PVB unit: nm^2. The ratio calculates the proportion between "w/ Diff-SO" and "w/o SO" of the corresponding MO algorithm.

differences in SO methods brought about by different learning models.

Diff-SO Effectiveness Evaluation: We conduct performance evaluations of Diff-SO with various MO algorithms [7], [12], [13], [14] in Table I and observe that combining any MO algorithm with the Diff-SO algorithm consistently improves L2 and PVB results. It is worth noting that in Table I, the "ratio" row represents the percentages calculated between results obtained using only MO (labeled as "w/o SO") and the results achieved by incorporating Diff-SO with the corresponding MO method within the Diff-SMO framework (labeled as "w/ Diff-SO"). When the vanilla MO algorithm MOSAIC [7] is combined with the Diff-SO algorithm, a significant reduction of 18% in L2 error and 7% in PVB is achieved. The SOTA level set-based MO algorithm, DevelSet [12], can reduce the L2 error by 12.5% and the PVB by 7.1% when combined with Diff-SO. When combined with A2ILT [13], the SOTA pixel-based MO algorithm, Diff-SO achieves a reduction of 11.8% in L2 error and 8.1% in PVB, respectively. The latest MO research, Multi-ILT [14], has made groundbreaking progress by utilizing multilevel ILT combined with SRAFs. However, Diff-SO still manages to achieve an overall reduction of 9.5% in L2 error and 7.8% in PVB. These experiments validate the effectiveness of Diff-SO within the Diff-SMO framework, demonstrating its capability to yield an average enhancement of approximately 13% in L2 error and 7.5% in PVB across various SOTA MO methods.

Diff-SMO Result Comparison With SOTA SMO Methods: As depicted in Figs. 8 and 9, the x-axis represents different MO algorithms. Each point represents a combination of different SMO algorithms. We compare the performance of Diff-SO with previous SOTA SO solvers, including ICC-CPU [17] and SoulNet [18]. Regarding MO solvers, unlike the original MOSAIC, our implementation entails GPU acceleration to enhance MOSAIC's computational efficiency. From Figs. 8 and 9, it can be observed that our approach, including GPU-SO and Diff-SO, outperforms the previous SOTA SO

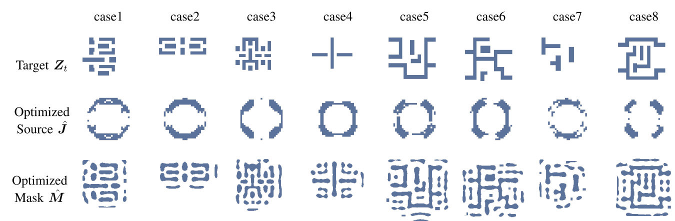

Fig. 8. L2 comparison of SMO algorithms.

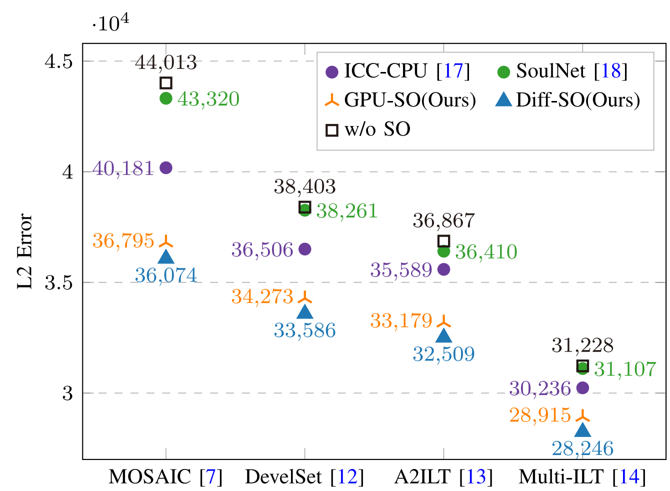

Fig. 9. PVB comparison of SMO algorithms.

methods in terms of both L2 and PVB metrics. Specifically, for Diff-SO, the L2 metric shows an average improvement of 9.2% and 14.3% compared to ICC-CPU and SoulNet, respectively. Regarding the PVB metric, Diff-SO demonstrates an average enhancement of 4.7% and 6.9% over ICC-CPU and SoulNet, respectively. It is noteworthy that the magnitude of MO optimization objectives directly influences the potential optimization space for Diff-SO. For example, when Multi-ILT is chosen as the MO solver, Diff-SO exhibits a modest 9.5% improvement in L2 alone. However, employing a less

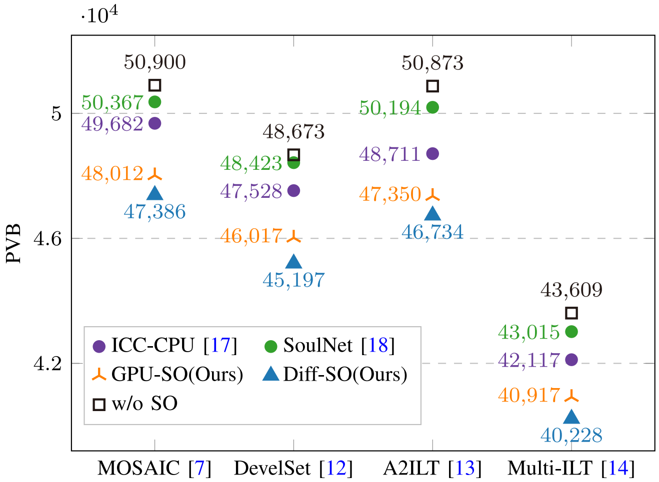

Fig. 10. Visualizations for Diff-SMO results with annular source as initial condition.

TABLE II
EPE COMPARISON

SO-solver	w/o SO	ICC-CPU [17]	SoulNet [18]	GPU-SO	Diff-SO
EPE	6.625	6.575	6.7	6.5	6.5
ratio	1.019	1.012	1.031	1	**1**

sophisticated MO solver, such as MOSAIC, allows Diff-SO to achieve a significant 18% enhancement in L2. Several sample results of Diff-SMO are depicted in Fig. 10, utilizing an annular source as the initial source condition.

EPE and Runtime Comparison of SMO Algorithms: In Tables II and III, we further compare the performance of EPE and runtime. The "average" in Table II for the row MO-solver represents the average results obtained by testing MOSAIC, DevelSet, A2ILT, and Multi-ILT as MO solvers. The table presents the average outcomes of these MO solvers when different SO solvers are used. Diff-SO achieves a reduction of 1.2% and 3.1% in EPE compared to ICC-CPU and SoulNet, respectively, as shown in Table II. Table III presents a comprehensive runtime comparison of the SMO algorithm execution, encompassing both the SO and MO phases. SoulNet [18] directly generates the source using an autoencoder, omitting the iterative SMO optimization process. Consequently, this leads to a modest 0.3-s increase in runtime for Diff-SMO compared to SoulNet. The observed increase can be attributed to the subsequent feeding of the generated source from the diffusion model into GPU-SO for further fine-grained optimization steps. However, the design has proven to be valuable, as the 0.3-s increase in runtime has resulted in a notable 14.3% decrease in L2 error. Moreover, in comparison to the conventional SO algorithm, ICC-CPU [17], Diff-SMO demonstrates a remarkable speed enhancement, achieving a nearly 225-fold improvement in the overall SMO runtime while producing better results.

SoulNet Versus Diffusion Model: In Table IV, we compare our diffusion model with SOTA learning-based method SoulNet [18]. The term "mIoU" stands for Mean Intersection over Union, which represents the average ratio of intersection

TABLE III
SMO RUNTIME COMPARISON

MO / SO	w/o SO	ICC-CPU [17]	SoulNet [18]	GPU-SO	Diff-SO
MOSAIC [7]	20.9*s*	1925*s*	21.7*s*	35.6*s*	21.9*s*
DevelSet [12]	1.4*s*	1863*s*	2.4*s*	17.5*s*	2.9*s*
A2ILT [13]	4.6*s*	1875*s*	5.6*s*	22.1*s*	5.9*s*
Multi-ILT [14]	1.3*s*	1869*s*	2.3*s*	16.9*s*	2.6*s*
SO Average	-	1883*s*	0.95*s*	16*s*	1.27*s*
SO Ratio	-	-	0.75	12.6	1
SMO Average	7.05*s*	1900*s*	8.00*s*	23.02*s*	8.32*s*
SMO Ratio	-	228.2	**0.96**	2.77	1

Time unit "*s*" refers to one second.

TABLE IV
SOULNET VERSUS OUR DIFFUSION MODEL

	mIoU	pixelAcc	inference time	Model
SoulNet [18]	93.86%	96.70%	0.09s	31MB
Diffusion	98.65%	99.14%	0.05s	30MB

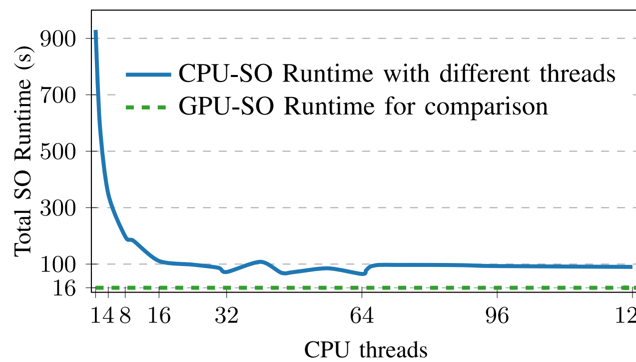

Fig. 11. Runtime performance of multicore CPU-based SO by using different threads is shown by blue solid line, and details can be found in Table V. The GPU-SO is also illustrated in green dashed line for comparison.

over union. "pixelAcc" represents pixel accuracy, defined as the percentage of pixels correctly classified in an image. Our diffusion model exhibits improvements of 4.79% in mIoU and 2.44% in pixelAcc. Additionally, it requires less inference time compared to SoulNet due to its simpler network architecture.

B. Ablation Study

CPU-SO Versus GPU-SO: We conducted a series of experiments to validate the efficiency of the GPU-accelerated SO algorithm. First, we compared it with the acceleration of SO using a multicore CPU with multithreads, referred to as CPU-SO. The CPU we used has 64 cores, supporting up to 128 threads. By varying the number of threads used (from 1 to 128), we averaged the CPU runtime for ten test cases, and the results are plotted in Fig. 11. The detailed experimental results for each sampled thread are shown in Table V. From Fig. 11 and Table V, it can be observed that between 1 to 32 threads, the runtime shows a significant decrease as the number of threads increases. When the number of threads exceeds 32, the runtime remains relatively stable. At 64 threads,

TABLE V
RUNTIME DETAILS OF SO USING MULTICORE CPUS

GPU-SO	CPU-SO with different threads							
Time(s)	Threads	1	2	4	8	10	16	24
	Time(s)	930	593	348	198	182	111	98
16.4	Threads	32	40	45	48	56	64	65
	Time(s)	72	108	69	72	85	**65**	78
	Threads	66	68	72	80	88	96	128
	Time(s)	91	97	97	97	96	93	90

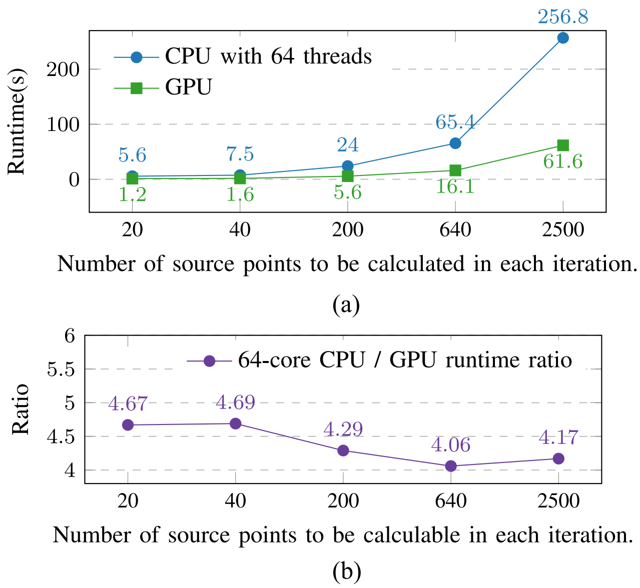

Fig. 12. By controlling the illumination area, and thus changing the number of effective light sources, we compare the runtime performance of a 64-thread CPU and a GPU. (a) Runtime performance when calculating different source points. (b) 64-thread CPU and GPU runtime ratio.

which is precisely the number of CPU cores, the best-runtime performance is achieved, taking 65 s to complete SO. In contrast, the GPU's runtime is 16 s. Notably, when the number of threads exceeds 64, even at 65, there's a noticeable increase in runtime. We believe this is due to CPU oversubscription, causing different processes to compete for CPU resources, resulting in reduced efficiency.

We also conducted another set of experiments. Given that the best-experimental results are achieved with 64 threads, how would the runtime for both the 64 threads CPU and GPU be affected if we change the number of source points that need to be optimized in each iteration, thereby altering the overall computational load. The experimental results are shown in Fig. 12. Observing (9), when ICC is fixed, the total computational load of SO is directly proportional to the source points of the illumination source $\boldsymbol{J}$. By adjusting the size of the source area, we controlled the number of source points to be computed, ranging from 20 to 2500. Now we can compare the runtime performance under varying computational loads to observe how CPU-SO and GPU-SO perform as the amount of computation changes. As seen in Fig. 12(a), as the number of effective source points increases, resulting in a larger computational load, the runtime required for SO increases. However, it's noteworthy that regardless of whether the number of source points is greater or less than 64, the ratio of runtime between the 64-thread CPU and GPU consistently remains around 4 times, as depicted in Fig. 12(b).

In summary, as shown in Fig. 11, when the computational workload is constant, we observe that the total runtime dramatically decreases with the increase in parallel threads, reaching a minimum at 64 threads. The runtime is approximately four times that of GPU-SO. Then, in Fig. 12, we vary the computational workload and compare the 64-threads CPU-based SO with the GPU-SO, maintaining a runtime that is roughly four times longer. This confirms the necessity and efficiency of our GPU-accelerated SO algorithm.

GPU-SO Versus Diff-SO: In Table III, we present a comparative analysis of runtime between GPU-SO and Diff-SO under the integration with various MO methods. It is observable that in the absence of the MO process, GPU-SO, on average, takes an additional 14.7 s to execute, approximately 12.6 times that of Diff-SO. By utilizing a diffusion model to generate near-optimal sources as initial inputs for GPU-SO iterations, Diff-SO significantly curtails the SO time by 92%, leading to enhanced optimization outcomes. From Figs. 8 and 9, it is evident that compared to GPU-SO, Diff-SO also augments the overall SMO performance, reducing the L2 error by 2% and the PVB by 1.5%. Relative to MO, the diffusion model is exceptionally suitable for SO. Typically, a mask clip handled is of size 2048 × 2048 pixels, and employing the diffusion model for such dimensions necessitates substantial memory and entails extensive computation time. In contrast, common source sizes range from 35 × 35 to 60 × 60. For sources of these dimensions, a modest GPU can leverage the diffusion model to generate sources with remarkable precision.

V. CONCLUSION

We introduce Diff-SMO, an advanced and accelerated solution for SMO. Our framework innovatively expands on existing MO algorithms, with a primary focus on accelerating and refining SO. Our distinctive contribution is the formulation of a GPU-accelerated algorithm grounded in Abbe's imaging theory, which enables full GPU acceleration throughout the SMO flow. Moreover, we integrate a novel diffusion model that gradually learns to activate or deactivate source points, resulting in rapid generation of approximately optimal solutions, significantly boosting the SMO process. Experimental outcomes validate the superiority of Diff-SMO over combined prior SOTA SO and MO algorithms. Remarkably, it accomplishes an acceleration exceeding 200 times compared to conventional SMO methods, concurrently producing superior results. In future work, we plan to open-source our lithography model and SMO framework, fostering further research in this area.

REFERENCES

[1] A. E. Rosenbluth et al., "Optimum mask and source patterns to print a given shape," *J. Micro/Nanolithogr., MEMS, MOEMS*, vol. 1, no. 1, pp. 13–30, 2002.

[2] X. Ma, L. Dong, C. Han, J. Gao, Y. Li, and G. R. Arce, "Gradient-based joint source polarization mask optimization for optical lithography," *J. Micro/Nanolithogr., MEMS, MOEMS*, vol. 14, no. 2, 2015, Art. no. 023504.

[3] J.-C. Yu and P. Yu, "Gradient-based fast source mask optimization (SMO)," in *Proc. 24th SPIE*, 2011, pp. 1–13.

[4] G. Chen, Z. Pei, H. Yang, Y. Ma, B. Yu, and M. Wong, "Physics-informed optical kernel regression using complex-valued neural fields," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 2023, pp. 1–6.

[5] H. H. Hopkins, "On the diffraction theory of optical images," *Proc. Royal Soc. London. Series A. Math. Phys. Sci.*, vol. 217, no. 1130, pp. 408–432, 1953.

[6] N. Cobb, *Sum of Coherent Systems Decomposition by SVD*, Univ. California, Berkeley, CA, USA, 1995.

[7] J.-R. Gao, X. Xu, B. Yu, and D. Z. Pan, "MOSAIC: Mask optimizing solution with process window aware inverse correction," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 2014, pp. 1–6.

[8] G. Chen, W. Chen, Q. Sun, Y. Ma, H. Yang, and B. Yu, "DAMO: Deep agile mask optimization for full-chip scale," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 41, no. 9, pp. 3118–3131, Sep. 2022.

[9] Z. Yu, G. Chen, Y. Ma, and B. Yu, "A GPU-enabled level-set method for mask optimization," *IEEE Trans. Comput. Aided Design Integr. Circuits Syst. (TCAD)*, vol. 42, no. 2, pp. 594–605, Feb. 2023.

[10] B. Zhu et al., "L2O-ILT: Learning to optimize inverse lithography techniques," *IEEE Trans. Comput. Aided Design Integr. Circuits Syst. (TCAD)*, early access, Oct. 10, 2023, doi: 10.1109/TCAD.2023.3323164.

[11] H. Yang, S. Li, Y. Ma, B. Yu, and E. F. Young, "GAN-OPC: Mask optimization with lithography-guided generative adversarial nets," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 2018, pp. 1–6.

[12] G. Chen, Z. Yu, H. Liu, Y. Ma, and B. Yu, "DevelSet: Deep neural level set for instant mask optimization," in *Proc. IEEE/ACM Int. Conf. Comput. Aided Design (ICCAD)*, 2021, pp. 1–9.

[13] Q. Wang, B. Jiang, M. D. F. Wong, and E. F. Y. Young, "A2-ILT: GPU accelerated Ilt with spatial attention mechanism," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 2022, pp. 967–972.

[14] S. Sun, F. Yang, B. Yu, L. Shang, and X. Zeng, "Efficient ILT via multi-level lithography simulation," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 2023, pp. 1–6.

[15] S. Banerjee, Z. Li, and S. R. Nassif, "ICCAD-2013 CAD contest in mask optimization and benchmark suite," in *Proc. IEEE/ACM Int. Conf. Comput. Aided Design (ICCAD)*, 2013, pp. 271–274.

[16] Z. Wang, X. Ma, R. Chen, S. Zhang, and G. R. Arce, "Fast pixelated lithographic source and mask joint optimization based on compressive sensing," *IEEE Trans. Comput. Imag.*, vol. 6, no. 1, pp. 981–992, Jun. 2020.

[17] Y. Sun, Y. Li, G. Liao, M. Yuan, P. Wei, Y. Li, L. Zou, and L. Liu, "Sampling-based imaging model for fast source and mask optimization in immersion lithography," *Appl. Opt.*, vol. 61, no. 2, pp. 523–531, 2022.

[18] Y. Chen et al., "SoulNet: Ultrafast optical source optimization utilizing generative neural networks for advanced lithography," *J. Micro/Nanolithogr., MEMS, MOEMS*, vol. 18, no. 4, pp. 1–11, 2019.

[19] P. Evanschitzky, A. Erdmann, and T. Fuehner, "Extended abbe approach for fast and accurate lithography imaging simulations," in *Proc. Eur. Mask Lithogr. Conf.*, 2009, pp. 1–11.

[20] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," in *Proc. 34th Conf. Neural Inf. Process. Syst.*, 2020, pp. 1–12.

[21] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli, "Deep unsupervised learning using nonequilibrium thermodynamics," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2015.

[22] J. Austin, D. D. Johnson, J. Ho, D. Tarlow, and R. van den Berg, "Structured denoising diffusion models in discrete state-spaces," in *Proc. 35th Conf. Neural Inf. Process. Syst.*, 2021, pp. 1–13.

[23] A. Vaswani et al., "Attention is all you need," in *Proc. Annu. Conf. Neural Inf. Process. Syst. (NeurIPS)*, 2017, pp. 1–11.

Guojin Chen received the B.Eng. degree in software engineering from Huazhong University of Science and Technology, Wuhan, China, in 2019. He is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong.

His current research interests include machine learning in VLSI design for manufacturability and physics-informed networks for solving EDA area problems.

Zixiao Wang received the B.Eng. degree in automation and the M.Sc. degree in computer science from Tsinghua University, Beijing, China, in 2019 and 2022, respectively. He is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong.

His research interests include generative AI × EDA.

Bei Yu (Senior Member, IEEE) received the Ph.D. degree from The University of Texas at Austin, Austin, TX, USA, in 2014.

He is currently an Associate Professor with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong.

Dr. Yu received ten Best Paper Awards from IEEE TSM 2022, DATE 2022, ICCAD 2021 and 2013, ASPDAC 2021 and 2012, ICTAI 2019, Integration, the VLSI Journal in 2018, ISPD 2017, SPIE Advanced Lithography Conference 2016, and six ICCAD/ISPD contest awards. He has served as a TPC Chair for ACM/IEEE Workshop on Machine Learning for CAD and in many journal editorial boards and conference committees. He is the Editor of IEEE TCCPS Newsletter.

David Z. Pan (Fellow, IEEE) received the B.S. degree from Peking University, Beijing, China, in 1992, and the M.S. and Ph.D. degrees from the University of California at Los Angeles, Los Angeles, CA, USA, in 1994, 1998, and 2000, respectively.

From 2000 to 2003, he was a Research Staff Member with IBM T. J. Watson Research Center, Yorktown Heights, NY, USA. He is currently a Silicon Labs Endowed Chair Professor with the Chandra Department of Electrical and Computer Engineering, The University of Texas at Austin, Austin, TX, USA. He has published over 480 refereed journal/conference papers and nine US patents. His research interests include electronic design automation, synergistic AI and IC co-optimizations, design for manufacturing, and design/CAD for analog/mixed-signal and emerging technologies.

Dr. Pan has received many awards, including 20 Best Paper Awards, the SRC Technical Excellence Award, the DAC Top 10 Author Award in Fifth Decade, and the ASP-DAC Frequently Cited Author Award, among others. He has served in many editorial boards and conference committees, including various leadership roles, such as the DAC 2024 Technical Program Chair, the ICCAD 2019 General Chair, and the ISPD 2008 General Chair. He is a Fellow of ACM and SPIE.

Martin D. F. Wong (Fellow, IEEE) received the B.Sc. degree in math from the University of Toronto, Toronto, ON, Canada, in 1979, and the M.S. degree in math and the Ph.D. degree in CS from the University of Illinois at Urbana-Champaign (UIUC), Champaign, IL, USA, in 1981 and 1987, respectively.

He was a Bruton Centennial Professor of CS with The University of Texas at Austin, Austin, TX, USA, and an Edward C. Jordan Professor of ECE with UIUC. From August 2012 to December 2018, he was the Executive Associate Dean of the College of Engineering, UIUC. From January 2019 to August 2023, he was the Dean of Engineering and the Choh-Ming Li Professor of Computer Science and Engineering with The Chinese University of Hong Kong (CUHK), Hong Kong. Since August 2023, he was with Hong Kong Baptist University (HKBU), Hong Kong, as the Provost of HKBU and a Chair Professor of Computer Science. He has published around 500 papers and graduated over 50 Ph.D. students in EDA. His main research interest is in electronic design automation (EDA).

Dr. Wong is a Fellow of ACM.