# FuILT: Full Chip ILT System With Boundary Healing

Shuo Yin[†]
The Chinese University of Hong Kong
Hong Kong SAR

Wenqian Zhao[†]
The Chinese University of Hong Kong
Hong Kong SAR

Li Xie
Shenzhen GWX Technology Co., Ltd.
China

Hong Chen[*]
Shenzhen GWX Technology Co., Ltd.
China

Yuzhe Ma
Hong Kong University of Science and Technology (Guangzhou)
China

Tsung-Yi Ho
The Chinese University of Hong Kong
Hong Kong SAR

Bei Yu[*]
The Chinese University of Hong Kong
Hong Kong SAR

## ABSTRACT

Mask optimization in lithography is becoming increasingly important as the technology node size shrinks down. Inverse Lithography Technology (ILT) is one of the most performant and robust solutions widely used in the industry, yet it still suffers from heavy time consumption and complexity. As the number of transistors scales up, the industry currently focuses more on efficiency improvement and workload distribution. Meanwhile, most recent publications are still tangled in local pattern restoration regardless of real manufacturing conditions. We are trying to extend academia to some real industrial bottlenecks with FuILT, a practical full-chip ILT-based mask optimization flow. Firstly, we build a multi-level partitioning strategy with the divide-and-conquer mindset to tackle the full-chip ILT problem. Secondly, we implement a workload distribution framework to maintain hardware efficiency with scalable multi-GPU parallelism. Thirdly, we propose a gradient-fusion technique and a multi-level healing strategy to fix the boundary error at different levels. Our experimental results on different layers from real designs show that FuILT is both effective and generalizable.

## CCS CONCEPTS

• **Hardware → VLSI design manufacturing considerations**.

## KEYWORDS

OPC, inverse lithography technology (ILT), full-chip

**ACM Reference Format:**
Shuo Yin[†], Wenqian Zhao[†], Li Xie, Hong Chen[*], Yuzhe Ma, Tsung-Yi Ho, and Bei Yu[*]. 2024. FuILT: Full Chip ILT System With Boundary Healing . In *Proceedings of the 2024 International Symposium on Physical Design (ISPD '24), March 12–15, 2024, Taipei, Taiwan.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3626184.3633315

## 1 INTRODUCTION

With the fast development of the semiconductor manufacturing process, the VLSI technology node has been shrinking rapidly. In lithography, it is inevitable to apply some resolution enhancement technologies (RET) to correct the optical proximity and process bias/effects. Many methods were proposed to enhance the lithography resilience to manufacturing variation, from the earliest rule-based OPC [1–5] to the latest machine learning (ML)-based [6–9] solutions. Unfortunately, these approaches either suffer from exploding solution space in advanced technology nodes or lack robustness to complicated critical patterns. Over the past three decades, inverse lithography technology (ILT) has become the dominating OPC solution with rigorous performance in real industry scenarios.

It is commonly accepted by academia and industry that ILT technologies still have the potential to be further explored. However, the focus of some recent research works might deviate from real manufacturing needs. Some research, such as [10–15] have put forward new ILT algorithms mainly focalizing on further optimizing masks of some small tiles. Nevertheless, how to extend the progress to full-chip scale for real design is often ignored. In such a scenario, these works can only tackle the whole design by slicing it into a stack of fix-size tiles and stitching back after optimizing each tile individually. However, This can be erroneous as the lithography effect on the slicing boundary was partially ignored when optimizing these tiles one by one. In fact, the full-chip design ILT is much more complex as the design size increase to a gigantic scale.

Recently, some academic works started to propose solutions for full-chip level mask optimization. For example, [16] and [17] both proposed to increase the efficiency of full-chip mask optimization via layers with a new partitioning method and adaptive solver selection. [8] added overlapping area in lithography to cover more neighboring information. However, the stitching error problem on the boundary still remains unresolved. Therefore, these approaches will face obstacles to extending to other layers with complicated patterns, such as metal layers.

[*]Corresponding authors, [†]Co-first authors

On the other hand, the industry tackled the problem from different perspectives. Identical to previously mentioned academic works, Intel's ILT solution [18] splits the full-chip design into small tiles and distributes each tile to different CPUs for a separate ILT process. After finishing the optimization of each tile, it uses a "stitch and heal" technique to reoptimize the region close to the stitching boundary. TrueMask [19] is the first commercial tool that can optimize full-chip designs within a day while enabling GPU acceleration. In order to support super large full-chip data at once, D2S built an ILT-specific emulated giant CPU/GPU pair to bear the processing pressure of full-chip mask optimization all at once.

In this paper, we present FuILT, a new open-source full-chip ILT-based mask optimization system that we implemented from scratch as a complete solution to tackle hardcore obstacles such as complexity explosion and stitching errors in conventional mask optimization processes. Note that ILT-based mask optimization treats patterns as pixelated images; the main obstacle we try to conquer is the large size of the real design. As the pixelated mask scale to the trillion level, the memory and computation bound of hardware rule out the possibility of deploying full-chip optimization all at once on a single machine.

We design a hierarchical mask partitioning strategy to recursively break full-chip patterns into areas of a certain size suitable for specific hardware devices. With a divide-and-conquer fashion to handle sliced tiles, we manage to optimize various areas in parallel. Although previous works have also applied partitioning strategies to handle large patterns, our hierarchical partitioning approach aims to address both memory and computation constraints separately. We have employed the Server-Worker strategy for distributed optimization. We can increase parallelism by plugging more GPUs on a single system or allocate the workload to a distributed system to ease memory pressure and maintain hardware efficiency.

Our framework also comprises of two boundary healing mechanisms at each partitioning level to tackle the stitching error or defects on the boundary. We propose a new gradient-fusion technology to smooth the boundary, rendering the ILT area from small tiles to a large pattern, regardless of hardware limitations. Our proposed full-chip ILT system is not dependent on any specific ILT algorithm. We can easily integrate any state-of-the-art ILT algorithm into our system by designing a new API. Moreover, we integrate a deep-learning toolkit in our framework for auto-gradient derivation with Multi-GPU/System parallelism. We can realize significant CPU-GPU acceleration with the pipelining of multi-thread and multi-process. The key contributions are summarized as follows:

- We are the first in academia to discuss how to solve the boundary stitching error in full-chip ILT, addressing the existing gap between academia and real industrial needs.
- We build up a practical full-chip ILT mask optimization system with efficiency enhancement and multi-level parallelism for real design.
- We propose a gradient-fusion technology along with a multi-level boundary healing strategy to solve the stitching error of large patterns in the real scenario.
- We verify our framework on different layers from real design to show the effectivity and generalizability.

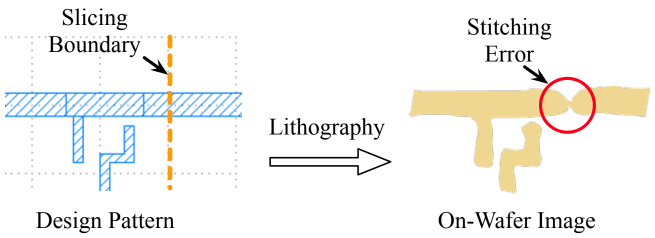


**Figure 1: Visualization of lithography defects after mask stitching. The printed image on the right shows disconnection occurring on the slicing boundary.**

The remainder of this paper is organized as follows. In Section 2, we introduce lithography defects and limitations, as well as the problem formulation. Section 3 presents our multi-level partition and stitching strategy, along with the allocation of workload in a Multi-GPU system. In Section 4, we present the experimental results at the full chip scale and compare them to the state-of-the-art. Finally, we conclude our paper in Section 5.

## 2 PRELIMINARIES

### 2.1 Lithography Defects and Limitations

In lithography, the optical projection system determines the minimum optically resolvable linewidth and depth of focus, which is restricted by the light wavelength and numerical aperture of the optical system. Despite the technological improvement of the optical system, such limits result in optical proximity and unavoidable bias. In addition to the diffraction effect during projection, the printed image is inevitably distorted from the original geometrical shape. Lithography defects such as unexpected pattern disconnections or collisions will jeopardize the chip functionality.

Apart from the mentioned errors from the physical process, other defects come from the engineering process. Normally, slicing a large design into a grid of small tiles is the very first step. It is not hard to notice that slicing will lose information on the boundary and incur more defects after stitching back, as visualized in Figure 1. Previous work such as [8] considered overlapping buffer regions to handle neighboring tiles for lithography, which partially eases the problem. On the other hand, the mask stitching process remains unresolved. Boundary errors such as mismatches or breakpoints need to be healed.

### 2.2 Inverse Lithography Technology

Inverse Lithography Technology (ILT) aims to acquire the optimal mask shape prior to lithography to etch the desired image on the wafer. By mathematically inversing the lithography process, ILT manipulates the adjustment and distortion of mask shape to compensate for the optical diffraction or process effects.

ILT comprises two major components: forward lithography simulation and inverse-lithography mask optimization. The forward lithography $f(\cdot)$ covers the complete mask-to-wafer process, including the optical projection and the photoresist process. It takes $\boldsymbol{M}$ as input and directly render the on-wafer image $\boldsymbol{Z}$, which can be formulated as:

$$\boldsymbol{Z} = f(\boldsymbol{M}). \quad (1)$$

Both mask $\boldsymbol{M}$ and image $\boldsymbol{Z}$ are binary 2-D matrices. The objective of the ILT process is to find the optimal mask $\boldsymbol{M}^*$ given the target image $\boldsymbol{Z}_{target}$ such that:

$$\boldsymbol{M}^* = f^{-1}(\boldsymbol{Z}_{target}). \tag{2}$$

The lithography process $f(\cdot)$ is ill-posed. For each target pattern $\boldsymbol{Z}_{target}$, there may exist zero or many matched masks $\boldsymbol{M}^*$. It is impossible to find a white-box well-defined inverse function $f^{-1}(\cdot)$. Therefore, many approaches get away with this problem by switching to directly optimizing the mask in a gradient-descent style, where the gradient on the mask is iteratively updated by gradient $\nabla_{\boldsymbol{M}}$ with the objective of minimizing the difference between etched pattern $\boldsymbol{Z}$ and target pattern $\boldsymbol{Z}_{target}$. By convention, the objective function is formulated as follows:

$$\min\{L(f(\boldsymbol{M}), \boldsymbol{Z}_{target}) + \alpha_{pvb} L_{pvb}(f_{pvb}(\boldsymbol{M}))\}, \tag{3}$$

where the loss $L$ measures how much the etched image distorts from the desired pattern. For example, the Frobenius norm $|\cdot|_F^2$ is a commonly used loss function in many SOTA ILT algorithms [11–13, 16]. $f_{pvb}$ denotes Process Variation Band (PVB) simulation while $L_{pvb}$ represents PVB loss, which is a common metric to evaluate the mask robustness to various incident light conditions, and $\alpha_{pvb}$ is the weight for $L_{pvb}$.

## 3 ALGORITHMS

In this section, we describe each stage of FuILT following the left-to-right order in Figure 2. First, we will introduce the multi-level layout partition strategy, including macro-level partition and micro-level partition, respectively. Then, we will discuss and describe our mask tile gradient calculation distributed strategy. Later, we verify and illustrate our macro-level stitching and boundary healing to get a full mask at the end.

### 3.1 Multi-level Partition

As VLSI design grows into an incredibly large scale, mask optimization of the whole design on one machine is nearly impossible. In fact, our experiment record shows that even the tiniest design layer with merely 1830 standard cells will cost more than 90 gigabytes of memory consumption during the ILT process. Both the memory bound and computation bound of the machine will hinder the ILT process efficiency during optimization. In Section 2.1, we introduced the need to consider the mutual influence between neighboring tiles during the lithography process when partitioning the layout into tiles. According to the above elaboration, we build our framework to ease the heavy burden of resource consumption and memory congestion from the macro-level to the micro-level while considering the boundary effect in the lithography process.

**Macro-level Partition**. On top of parallelism, our hierarchical partitioning strategy will handle memory bound first by cutting the whole design layer into grids of large patches. Note that such partitioning is different from conventional trials in terms of granularity. Macro-level partitioning determines the number of patches based on the maximum storage memory of the system rather than the maximum computational memory. This approach generates a coarser-grained partition. Apparently, we do not have unlimited CPU/GPU resources on a single machine as commercial company D2S claimed. Budget and flexibility are our concerns as well because we are determined to reach a cost-effective design. As visualized in the left part of Figure 2, we aggressively slice the full-chip into partitions as large as possible so that boundary length and potential stitching area are minimized, which is formulated in Equation (4):

$$\begin{aligned} m^*, n^* = \underset{m,n}{\operatorname{argmin}} & \sum_{i}^{m\times n} (H_{\boldsymbol{P}_i} + W_{\boldsymbol{P}_i}), \\ \text{s.t.} \quad & \sum_{i}^{m} H_{\boldsymbol{P}_i} < H_{\boldsymbol{P}} + H'_{max}, \\ & \sum_{i}^{n} W_{\boldsymbol{P}_i} < W_{\boldsymbol{P}} + W'_{max}, \\ & Size(\boldsymbol{P}_i) + Size(\boldsymbol{G}_i) \leq MemSize, \end{aligned} \tag{4}$$

where $\boldsymbol{P}$ is the full design pattern which is partitioned into $m^* \times n^*$ patches, $\boldsymbol{P}_i$ represents the $i$-th patch and $\boldsymbol{G}_i$ denotes the gradient of $i$-th patch, which we will discuss detailed in section 3.2. $H_{\boldsymbol{P}}/W_{\boldsymbol{P}}$ and $H_{\boldsymbol{P}_i}/W_{\boldsymbol{P}_i}$ denote height/width of the pattern and $i$-th patch. $H'_{max}/W'_{max}$ represent the max overlapping length in height and max overlapping length in width, respectively. In our implementation, the size of each partition in the grid is delicately chosen just so the storage usage of mask and corresponding gradient matrices will not crash the main memory capacity $MemSize$. After first partitioning, all patches are stacked into a task queue to be sequentially handled.

Mask optimization of each patch is handled independently so far, and our framework supports a multi-devices system to assign each CPU-GPU pair to different patches to process in parallel. Namely, more devices indicate more parallel processes in the solver pool. We denote such full-chip slicing as macro-level partitioning.

**Micro-level Partition**. At the micro-level, we start handling each patch while considering the computational capability of GPUs. Each GPU is composed of many Streaming multiprocessors (SM), while each Streaming multiprocessor can execute at most a certain number of thread blocks in parallel. Therefore, we need to limit the size to avoid threads competition which brings excessive time delay. Besides, due to the limited on-chip cache size of each SM, we also need to shrink the size further to avoid frequent cache-memory data movement. We slice each large patch into four tiles and repeat recursively until each tile satisfies the mentioned GPU computation limits. The reason we partition each patch in a recursive way is that we can deep fusion tiles gradient through each level to illuminate the boundary error. We will discuss it in Section 3.3. At each level, a tile is further sliced into a "⊞" grid of four smaller tiles with overlapping as shown in Figure 2. Each derived tile is around 1/4 of the size of the tile before slicing. After $k$ levels of recursion, we obtain a set of $4^k$ tiles defined as $\boldsymbol{\tau_k}$, and the final level $k^*$ is determined by the GPU capacity with minimized time overhead in the experiment. The height and width of tiles are then slightly large than 1/4 before slicing with an overlapping stripe of length $\frac{max(H,W)}{R}$. The complete recursive micro-level partitioning flow is illustrated in Algorithm 1. We also keep the overlapping rate $R$ on the tile boundary for stitching error healing in the next section.

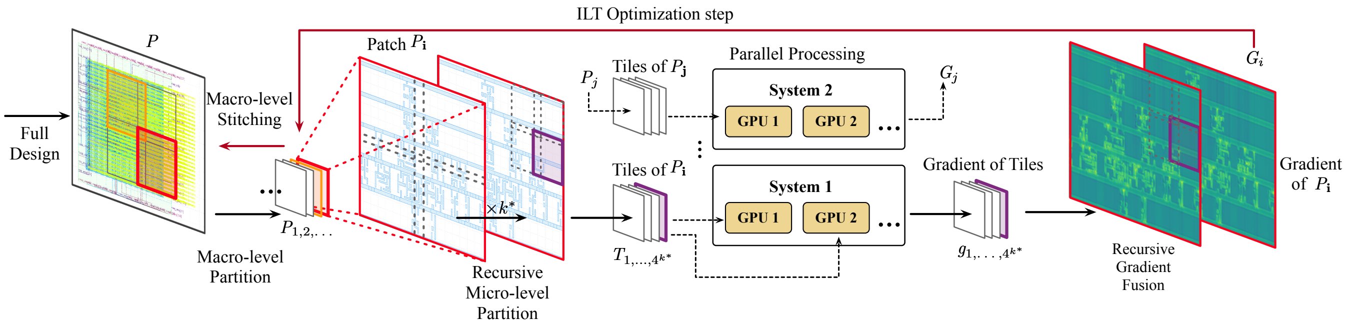


**Figure 2: The overall workflow of FuILT.**

**Algorithm 1** Micro-level Partition `Partition(P, k, R)`

```
Input: Patch $P$, Recursion level $k$, Overlapping rate $R$.
Output: All tiles sliced at $k$-th level $\tau_k, k \in [1, ..., k^*]$.
1: Initial: $k \leftarrow 1$, $\tau_0 = P$
2: if $k > k^*$ then
3:     return
4: end if
5: $H, W \leftarrow$ height and width of $P$
6: $s_h \leftarrow \lceil \frac{max(H,W)}{R} + H/2 \rceil$, $s_w \leftarrow \lceil \frac{max(H,W)}{R} + W/2 \rceil$ ▷ Compute slicing tile size at k-th level
7: $T_1,...,T_4 \leftarrow$ Slicing $P$ as $\begin{bmatrix} T_{1[s_h \times s_w]} & T_{3[s_h \times s_w]} \\ T_{2[s_h \times s_w]} & T_{4[s_h \times s_w]} \end{bmatrix}$
8: for $i \in [1, ..., 4]$ do
9:     $\tau_k \leftarrow \tau_k \cup T_i$
10:    Partition($T_i, k+1, R$)          ▷ Recurse next level
11: end for
```

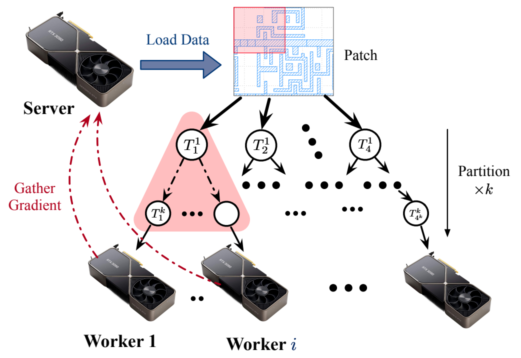


**Figure 3: Illustration of micro-level partition tree and the Server-Worker strategy of multi-GPU system.**

## 3.2 Multi-GPU Gradient Calculation

After finishing the micro-level partitioning, we calculate the gradient $g_i \in g[1, ..., 4^{k^*}]$ of each small tile $T_i \in \tau_{k^*}, i \in [1, ..., 4^{k^*}]$.

**Multi-GPU Pipeline.** To adapt the overall tiles workload $\tau_{k^*}$ to a Multi-GPU system, we utilize the Server-Worker strategy as shown in Figure 3. We will designate the first GPU as the `server` since we plan to use it for stitching the tiles gradients in Section 3.3, while the remaining GPUs will serve as `workers`. The role of `server` is to distribute the workload $\mathcal{T}_d \subset \tau_{k^*}$ to worker $d$ and gather all tile gradients from them. On the other hand, `workers` are responsible for computing the tile gradient. We compute the tile gradient with

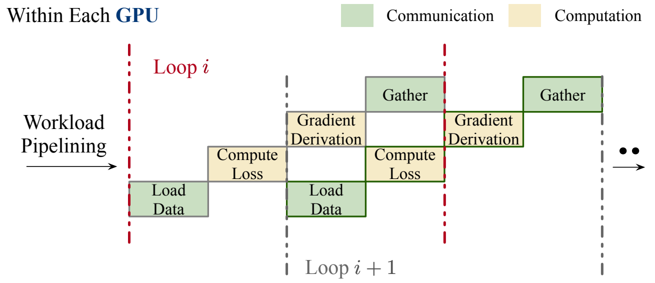


**Figure 4: Workload pipeline within each worker GPU.**

the help of deep learning toolkit `PyTorch` since it provides well-defined APIs for forward/backward calls with an automatic gradient engine and easy Multi-GPU extension.

From Figure 4, we can see that the workflow consists of four stages: each worker loads a tile from the `server`, performs ILT loss computing (forward) and gradient derivation (backward) operations, and finally gathers the gradient back to `server`. Given that the ILT process is individual for each tile $T_i$, we can organize the workflow into a two-stage pipeline to reduce communication costs between the `server` and `workers`. For the first stage of the pipeline, we adopt the ILT backward process and load the next tile $T_{next}$ and its corresponding target image $Z_{next}$ in the same time from the `server` and global system memory, respectively. In the second stage of the pipeline, we begin by performing the ILT forward process on the current tile $T$ while simultaneously gathering the gradient $g$ of the previous tile $T_{pre}$. The Algorithm 2 shows the detailed pseudocode of the Multi-GPU pipeline algorithm.

## 3.3 Multi-level Stitching & Boundary Healing

In correspondence to multi-level partitioning, we also apply multi-level boundary error healing to tackle defects mentioned in Section 2.1 during stitching.

**Micro-level Stitching & Boundary Healing**. Once we have the gradients $g_{1,...,4^{k^*}}$ of all small tiles $T_{1,...,4^{k^*}}$ partitioned from large patch $P_i$, we will fuse the gradient matrices back to a large gradient map $G_i^{(t)}$ in the iteration $t$. The gradient of each tile is fused back to its relative location within $P_i$, as shown in the right part of Figure 2. As for the overlapping area, the printed image is affected by all surrounding tiles. We conjecture that the neighboring effect $\psi(\cdot)$ of in lithography process is a linear combination of influence from every single object $p \in P_i$ in the neighboring area. Hypothetically,

**Algorithm 2** Multi-GPU Pipeline `Pipeline` $(\mathcal{T}_d,$ `ILT`$)$

```
Input: Workload $\mathcal{T}_d$ for worker $d$, certain ILT algorithm.
Output: Gradients of Workload $\mathcal{T}_d$.
 1: Initial: $T_{pre}, Z_{pre}, T_{next}, Z_{next}$
 2: $T_{pre}, Z_{pre} \leftarrow load(\mathcal{T}_d[1])$    ▷ load tile $T$ and target image $Z$ from server and global memory
 3: for $i \in [2, ..., |\mathcal{T}_d|]$ do
 4:     $loss \leftarrow$ ILT$(T_{pre}, Z_{pre})$
 5:     $T_{next}, Z_{next} \leftarrow load(\mathcal{T}_d[i])$
 6:     $g \leftarrow$ backward$(loss)$    ▷ one time ILT backward, which can run concurrently with load
 7:     gather$(g)$    ▷ send the tile gradient to server device, which can run concurrently with ILT
 8:     $T_{pre}, Z_{pre} \leftarrow T_{next}, Z_{next}$    ▷ change buffer state
 9: end for
10: $loss \leftarrow$ ILT$(T_{pre}, Z_{pre})$
11: $g \leftarrow$ backward$(loss)$    ▷ get final tile gradient
12: gather$(g)$
13: return
```

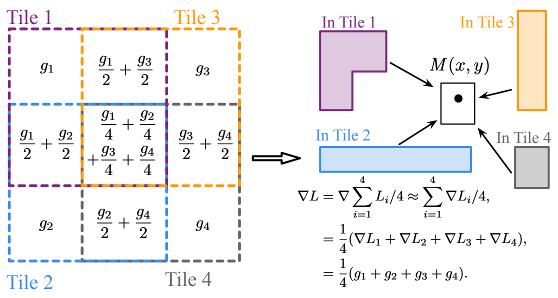

Tile 1
Tile 3
Tile 2
Tile 4
$g_1$
$\frac{g_1}{2}+\frac{g_3}{2}$
$g_3$
$\frac{g_1}{2}+\frac{g_2}{2}$
$\frac{g_1}{4}+\frac{g_2}{4}+\frac{g_3}{4}+\frac{g_4}{4}$
$\frac{g_3}{2}+\frac{g_4}{2}$
$g_2$
$\frac{g_2}{2}+\frac{g_4}{2}$
$g_4$
In Tile 1
In Tile 3
In Tile 2
In Tile 4
$M(x,y)$
$\nabla L = \nabla\sum_{i=1}^{4} L_i/4 \approx \sum_{i=1}^{4}\nabla L_i/4,$
$= \frac{1}{4}(\nabla L_1+\nabla L_2+\nabla L_3+\nabla L_4),$
$= \frac{1}{4}(g_1+g_2+g_3+g_4).$

**Figure 5: Gradient fusion on the overlapping area.**

we can infer based on the conjecture that the influence resulting in the final lithography loss $L$ is linear as well if the conjecture holds, and the gradient $\frac{\partial L}{\partial P_i}$ as well:

$$\begin{aligned}
&\psi(p_1, p_2, ...) \approx \sum_i \alpha_i \psi(p_i), \\
\Longleftrightarrow\ & L(\psi(p_1, p_2, ...)) \approx \sum_i L(\alpha_i \psi(p_i)), \\
\Longleftrightarrow\ & \frac{\partial}{\partial t} L(\psi(p_1, p_2, ...)) \approx \frac{\partial}{\partial t} \sum_i \alpha_i L(\psi(p_i)),
\end{aligned} \tag{5}$$

where $\alpha_i$ is the weight for the influnce of each neighboring object $\psi(p_i)$. As shown in Figure 5 we fuse the gradient of each pixel $(x, y)$ of the overlapping area by linearly adding the gradient of each neighboring tile. We apply gradient fusion recursively, following the pattern stitching at each level. We get $G_i^{(t)}$ with the same size of $\boldsymbol{P_i}$ at the end. The complete gradient fusion of our implementation is illustrated in Algorithm 3. Therefore, we can directly update large patch $\boldsymbol{P}$ with a fused large gradient map instead of a stack of small tiles. The advantage of gradient fusion is that: We ensure that the optimization process for the entire patch $\boldsymbol{P}$ is synchronized at each iteration and that the boundary region is updated in the same direction. So, the ILT process can be regarded with no partition at all. Defects such as mismatch or discontinuity on the boundary are cured in this process.

**Algorithm 3** Gradient Fusion `Fusion` $(\boldsymbol{g}[1, ..., n], k)$

```
Input: Recursive Level $k^*$, All tiles sliced at $k$-th level $\boldsymbol{\tau}_k, k \in [1, ..., k^*]$, $k^*$ level tiles gradient $\boldsymbol{g}[1, ..., 4^{k^*}]$
Output: The gradient $G$ of mask $P_i$
 1: Initial: $k \leftarrow k^*, n \leftarrow 4^{k^*}$
 2: $C \leftarrow \{\}$
 3: for $i \in [1, ..., 4^k]$ do
 4:     for $id \in [1, ..., 4^{k-1}]$ do    ▷ Cluster $g_i$ into groups
 5:         $tile \leftarrow \boldsymbol{\tau}_{k-1}[id]$
 6:         if $\boldsymbol{\tau}_k[i] \subset tile$ then
 7:             break
 8:         end if
 9:     end for
10:     $C[id] \leftarrow C[id] \cup g[i]$
11: end for
12: $\boldsymbol{g}' \leftarrow [\,]$
13: for $i \in [1, ..., \frac{n}{4}]$ do
14:     $\boldsymbol{g}'[i] \leftarrow$ Stitch$(C[i][1], C[i][2], C[i][3], C[i][4])$    ▷ Figure 5
15: end for
16: if $k == 1$ then
17:     $G \leftarrow \boldsymbol{g}'[1]$
        return $G$
18: else
        return Fusion $(\boldsymbol{g}'[1, ..., \frac{n}{4}],\ k-1)$    ▷ Recursive
19: end if
```

**Macro-level Stitching & Boundary Healing**. After obtaining the masks of large patches with inner boundaries fixed, we also need to tackle the stitching boundary of these large patches. Considering the huge patch size, we only apply a small healing box $M$ on the boundary area to heal stitching errors. The healing box slides along the boundary while running ILT steps on the overlapping area. For lithography forward, each restoration takes in the whole healing box as a forward area. Meanwhile, the backward step will only update the slim boundary.

All three conditions of macro-level healing are demonstrated in Figure 6. We only keep the gradient matrix on the boundary during each ILT optimization step to make the healing only occur on the boundary. We stitch all optimized patches with such macro-level healing techniques to achieve an original-size full-chip mask. Figure 7 is a visualization of the healing effect on a real metal-layer design. We can notice that the healing box and gradient masking will only fix the collision/misconnection area near the boundary but leave the outside area unchanged. This can significantly reduce the negative impact of new stitching boundary error generated by the healing box. For the macro-level healing box, We heuristically pick the overlapping width of the macro-level partition as the healing box size.

The complete flow follows a divide-and-conquer style. As shown in Figure 2, both micro-level and macro-level boundary healing are essentially the reverse process of macro-level and micro-level partitioning. Since FuILT is not sensitive to specific ILT algorithms, it can easily be extended to any state-of-the-art ILT algorithms.

## 4 EXPERIMENTAL RESULTS

### 4.1 Experimental Setup

FuILT is mainly developed in Python and `PyTorch`. All performance and speed evaluations are conducted on Ubuntu 20.04 with

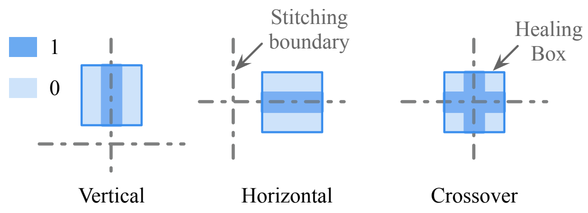


**Figure 6: Macro-level boundary healing strategy. The healing box is the ILT lithography area. We add a mask on the gradient $\frac{\partial L}{\partial M}$ during backward propagation. The dark blue area is 1 where the gradient is kept, and the light blue area is 0 where the gradient is neglected.**

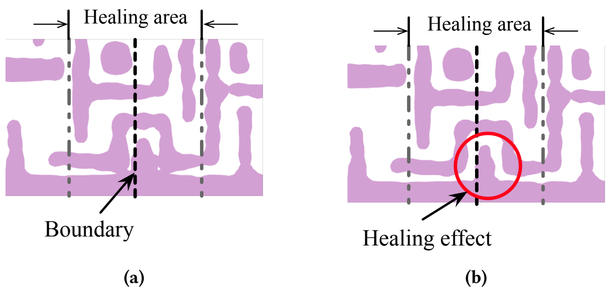


**Figure 7: Visualization of healing effect. The left figure is the overlapping area without macro-level healing. The right figure is the same area after macro-level healing. This crop is from our `Metal-1` layer result in section 4.**

**Table 1: Benchmark Details**

| Bench | #Polygons | Bounding Box ($nm \times nm$) | Total Area ($nm^2$) | Average Degree |
|---|---|---|---|---|
| Metal-1 | 5043 | 80528 × 80192 | 1675692925 | 5.8 |
| Via | 5411 | 73696 × 68992 | 22861474 | 4.0 |
| Poly | 1126 | 73696 × 68992 | 88904249 | 5.1 |
| Pimplant | 1830 | 80528 × 80192 | 3892569599 | 4.0 |

Intel(R) Xeon(R) Silver 4210R CPU @ 2.40GHz CPU (128G RAM) and two NVIDIA GeForce RTX 3090 GPU (24G). The photoresist intensity threshold is set at 0.225, and sigmoid steepness is 50. Our lithography process uses a real industry-level CPU-based simulator. The lithography wavelength is 193$nm$ with defocus range of ±30$nm$ and dose range of ±3%. The resolution is 28$nm$/pixel for the mask and 1$nm$/pixel for the target pattern/on-wafer image. The micro-level partition overlapping rate $R$ is set to 1/10. The ILT algorithm we use is Levelset-GPU[12] and $w_i^{(0)}$ is set to be 1.

For authenticity, the benchmark used in our experiments is from a real design which we extracted from a GDSII file generated by the open-source layout generation tool OpenROAD [20] and FreePDK45 [21] process design kit. We extracted four entire layers Metal-1, Via, Poly, and Pimplant with significantly different geometric shapes to show the generalizability. The details of the benchmark are provided in Table 1. In this table, #Polygons refers to the total number of polygons, and Average Degree denotes the average degree of all polygons.

In our implementation, each design is sliced into 4 patches at the macro level, given the main memory size limit. At the micro-level partitioning step, each patch is recursively sliced into 64 ($4^3$) small tiles to fit in the computation resources of each GPU.

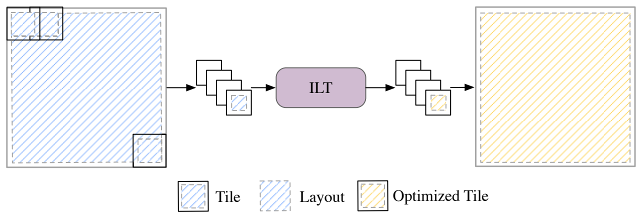


**Figure 8: The partition strategy of DAC'22 [9].**

All mask quality is simulated and evaluated at the full-chip scale. The measurements include Edge Placement Error (EPE) loss, Process Variability Band loss, $L_2$ loss, and total runtime. EPE violation threshold $th_{EPE}$ is set to 10$nm$. Our lithography simulator takes the stitched large mask as input and generates the full-layer pattern in one lithography step. This is consistent with the real scenario, so the results are more convincing.

## 4.2 Performance Analysis

We conduct our experiments in ablative order to verify the effectiveness of each module in our proposed framework. The baseline method follows the full-chip slicing strategy lately proposed in [9]. This slicing strategy has been widely adopted in previous full-chip OPC research [8, 9, 16, 17]. As shown in Figure 8, such a slicing strategy directly slices the layout into several small tiles with overlapping regions at once. After sequentially handling each small tile, it simply stitches the mask back directly without further modification. We compare this strategy as our baseline stitching method because this is the only full-chip OPC strategy proposed in academia. Therefore, we can demonstrate the significant improvements of our full-chip ILT system. For a fair comparison, we implement the exact same level-set method [12] as the baseline with the same parameters described in Section 4.1 for the gradient-descent update for both the baseline and FuILT. We extended the baseline to multi-GPU parallelism by simply scattering the mask to GPUs and gathering the optimized result.

Firstly and most importantly, we show the significant performance enhancement of our micro-level partition and gradient fusion technique. Both the baseline approach and our multi-level partitioning strategy slice the design into 16 × 16 partitions. Since the baseline strategy does not include boundary healing, we have excluded macro-level boundary healing for a fair comparison. The comparison results are presented in the first two chunks of Table 2. Our technique alone significantly reduces EPE violations by an average of 9 times across all four layers and improves L2 loss by 34% while also achieving a 31% PVBand improvement. The partitioning details are consistent with Section 4.1, which is our default framework setting for this design. Moreover, if we look at the most complicated Metal-1 layer, our technique achieves EPE reduction of up to 20 times and L2 loss reduction of 41%. Note that the more critical the pattern is, the more polygon breakdowns and stitching errors may occur. Such a big leap in error reduction sufficiently shows the effectiveness of our seamless boundary healing. Although our method requires more time than the baseline methodology due to the need to handle communication between GPU pairs during each

**Table 2: Comparisons of EPE, PVBand, $L_2$ Loss and Runtime**

| Bench | DAC'22[9] w/o. Macro Boundary Healing | | | | | FuILT w/o. Macro Boundary Healing | | | | | FuILT w. Macro Boundary Healing | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | #EPE | PVB ($nm^2$) | $L_2$ ($nm^2$) | RT (s) | S($\times 10^7$) | #EPE | PVB ($nm^2$) | $L_2$ ($nm^2$) | RT (s) | S($\times 10^7$) | #EPE | PVB ($nm^2$) | $L_2$ ($nm^2$) | RT (s) | S($\times 10^7$) |
| Metal-1 | 24668 | 120961048 | 224397927 | 2956 | 83.16 | 1243 | 66938717 | 158733917 | 3179 | 43.27 | 1221 | 66810974 | 154685729 | 4563 | 42.80 |
| Via | 127 | 6520380 | 9851354 | 1827 | 3.66 | 10 | 6445143 | 7931745 | 1902 | 3.38 | 10 | 6432513 | 7815421 | 2741 | 3.36 |
| Poly | 164 | 40238769 | 43742084 | 1831 | 20.55 | 59 | 32249748 | 31708247 | 1936 | 16.10 | 53 | 32055284 | 30297295 | 2836 | 15.88 |
| Pimplant | 4885 | 76507491 | 58102567 | 2837 | 38.86 | 1674 | 76389450 | 56955213 | 3174 | 37.09 | 1668 | 76310462 | 56664328 | 4475 | 37.03 |
| Total | 29844 | 244227688 | 336093932 | 9451 | 146.23 | 2986 | 182023058 | 255329122 | 10191 | 99.84 | 2953 | 181609233 | 249462773 | 14615 | 99.07 |
| Ratio | 9.99 | 1.34 | 1.31 | **1.00** | 1.46 | **1.00** | **1.00** | **1.00** | 1.08 | **1.00** | 0.98 | 0.99 | 0.97 | 1.54 | 0.99 |

optimization iteration, we believe that the sacrifice of an 8% increase in runtime is worth the significant improvements achieved.

Secondly, we validate that the macro-level boundary healing can further improves mask quality. In comparison, our experiment adds macro-level boundary healing to the optimized mask along the stitching boundaries, and the performance is shown in the third chunk of Table 2. Four benchmark layers show that macro-level boundary healing will reduce EPE errors by 2% on average and a slight reduction in PVBand and L2 loss. Although macro-level boundary healing may introduce new stitching boundary errors due to the healing box, its ability to slightly reduce errors provides evidence that boundary errors have been significantly mitigated and helps to compensate for the new stitching boundary errors introduced by the healing box.

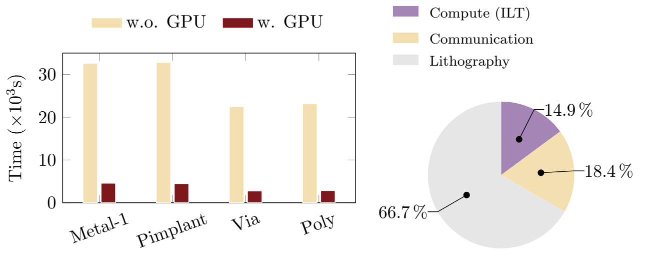
We also collect the overall score comparison in the fifth chunk of Table 2, where the score is the mask quality score measurement proposed by ICCAD Contest 2013[22] and "w. Healing" denotes approaches with macro-level boundary healing. So far, we do not have mask-checking tools to count violations on a full-chip pattern. Therefore, we replace the shape violations with $L_2$ loss and make it the weighted combination of the EPE, PVBand, Runtime, and $L_2$ loss. The modified score calculation is formulated as (6):

$$\text{Score} = \text{Runtime} + L_2 + 4 \times \text{PVBand} + 5000 \times \text{EPE}. \tag{6}$$

The score also shows that our approach achieves the optimal overall benchmarks.

## 4.3 Runtime Analysis

Apart from performance analysis, we also evaluate FuILT design in terms of time efficiency. As shown on the left of Figure 9, we demonstrate the acceleration ratio on all four benchmarks with our multi-GPU system. FuILT realizes ×7.6 speed-up on average for all four benchmarks above. We also visualize the time breakdown of all three parts of the complete flow: Lithography Simulation, CPU-GPU/GPU-GPU communication, and ILT Computation on the right figure of Figure 9. It is obvious that lithography itself occupies the majority of time consumption (around 67% of total time), given that our simulator is only licensed for CPU at this point. In other words, FuILT has already pushed to the limit of loss/gradient calculation and mask update (14.9% of total time) in the ILT process. The acceleration rate on ILT calculation itself is more than ×50, surprisingly.



**(a) Speed-up visualization with the multi-GPU mechanism.** **(b) Time breakdown of complete FuILT flow.**

**Figure 9: Time consumption analysis of FuILT.**

## 4.4 Visualization Result

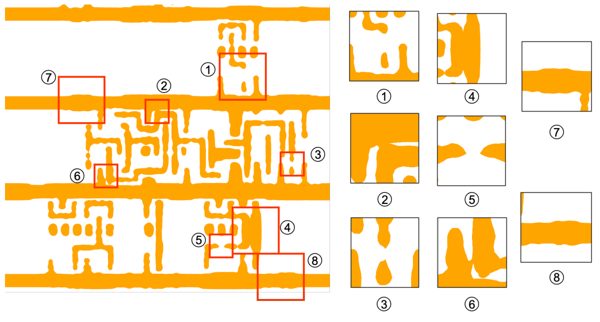
In addition to evaluation metrics, the visualization of the final printed images is equally important to demonstrate that FuILT effectively heals errors on the boundary.

As shown in Figure 10, we visualize printed images of both large-scale patterns and small local crops on the stitching boundary of Metal-1 layer. There are three types of errors that FuILT fixed. Crop-1, crop-3, and crop-5 are the unexpected disconnections of patterns on the printed image. On the other hand, crop-2, crop-4, and crop-6 show collisions and the emergence of objects where the original design is disjoint in these areas. These two types of errors are typical shape violations, which will seriously affect the functionality of the wafer image. Crop-7 and crop-8 are the typical distorted area when the original design is flat and straight where stitching incurs expected bulges. In comparison, our seamless boundary-healing strategy will solve all these stitching errors.

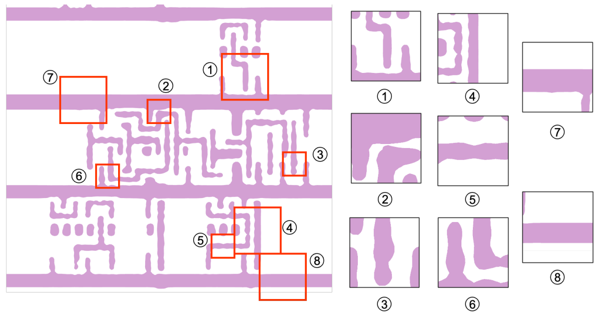
Furthermore, as shown in Figure 11, FuILT generates a result that is nearly identical to the target image, while the baseline method has numerous violations, with some via patterns even disappearing. These results demonstrate that even for the Via layer, the diffraction influence of neighboring tiles cannot be ignored.

## 5 CONCLUSION

In this paper, we proposed a brand new full-chip ILT system—FuILT with a multi-level boundary healing strategy and generalizability to all types of pattern layers with different geometric complexity. Our framework is capable of distributing the workload to multiple machines and multiple GPUs to realize multi-level parallelism. With a newly-proposed micro-level gradient fusion technique and macro-level stitching error healing, our framework can perform seamless boundary healing for any full-chip scale. Ablative experiments and

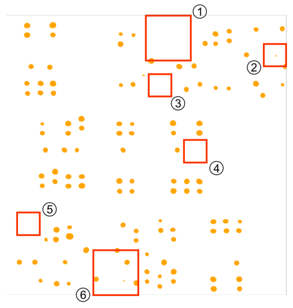


(a) DAC'22 [9]

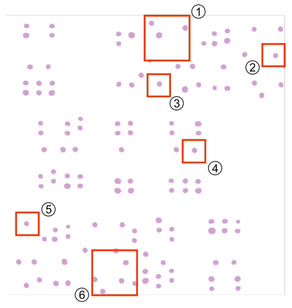


(b) FuILT

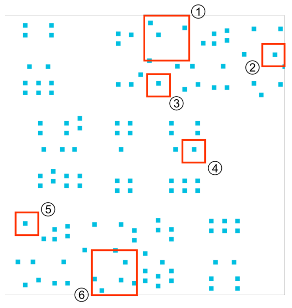
**Figure 10: Visualization of the Metal-1 layer printed image, showing both large-scale patterns and local boundary error areas. The figure displayed here is a small piece of the whole Metal-1 layer printed on-wafer image.**



(a) DAC'22 [9]



(b) FuILT



(c) Target Image

**Figure 11: Visualization of the Via layer printed image. The figure compares via optimization result with the target image.**

visualization show we can significantly eliminate pattern violations on the slicing boundary.

## ACKNOWLEDGEMENT

This work is partially supported by The Research Grants Council of Hong Kong SAR (No. CUHK14208021).

## REFERENCES

[1] O. W. Otto, J. G. Garofalo, K. K. Low *et al.*, "Automated optical proximity correction: a rules-based approach," in *Proc. SPIE*, 1994, pp. 278–293.
[2] P. Yu, S. X. Shi, and D. Z. Pan, "True process variation aware optical proximity correction with variational lithography modeling and model calibration," *JM3*, vol. 6, no. 3, pp. 031 004–031 004, 2007.
[3] X. Li, G. Luk-Pat, C. Cork, L. Barnes, and K. Lucas, "Double-patterning-friendly OPC," in *Proc. SPIE*, vol. 7274, 2009.
[4] Y.-H. Su, Y.-C. Huang, L.-C. Tsai, Y.-W. Chang, and S. Banerjee, "Fast lithographic mask optimization considering process variation," *IEEE TCAD*, vol. 35, no. 8, pp. 1345–1357, 2016.
[5] N. B. Cobb, "Fast optical and process proximity correction algorithms for integrated circuit manufacturing," Ph.D. dissertation, University of California at Berkeley, 1998.
[6] S. Choi, S. Shim, and Y. Shin, "Machine learning (ML)-guided OPC using basis functions of polar fourier transform," in *Proc. SPIE*, vol. 9780, 2016.
[7] 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.
[8] H. Yang and H. Ren, "Enabling scalable ai computational lithography with physics-inspired models," in *Proc. ASPDAC*, 2023, pp. 715–720.
[9] 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.
[10] 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, pp. 131:1–131:6.
[11] B. Jiang, L. Liu, Y. Ma, H. Zhang, E. F. Y. Young, and B. Yu, "Neural-ILT: Migrating ILT to nerual networks for mask printability and complexity co-optimizaton"," in *Proc. ICCAD*, 2020.
[12] Z. Yu, G. Chen, Y. Ma, and B. Yu, "A GPU-enabled level set method for mask optimization," in *Proc. DATE*, 2021.
[13] G. Chen, Z. Yu, H. Liu, Y. Ma, and B. Yu, "DevelSet: Deep neural level set for instant mask optimization," in *Proc. ICCAD*, 2021.
[14] T. Matsunawa, B. Yu, and D. Z. Pan, "Optical proximity correction with hierarchical bayes model," *JM3*, vol. 15, no. 2, p. 021009, 2016.
[15] 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.
[16] G. Chen, W. Chen, Y. Ma, H. Yang, and B. Yu, "DAMO: Deep agile mask optimization for full chip scale," in *Proc. ICCAD*, 2020.
[17] 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.
[18] V. Singh, B. Hu, K. Toh, S. Bollepalli, S. Wagner, and Y. Borodovsky, "Making a trillion pixels dance," in *Optical Microlithography XXI*, vol. 6924. SPIE, 2008, pp. 264–275.
[19] L. L. Pang, P. J. Ungar, A. Bouaricha, L. Sha, M. Pomerantsev, M. Niewczas, K. Wang, B. Su, R. Pearman, and A. Fujimura, "Truemask ilt mwco: full-chip curvilinear ilt in a day and full mask multi-beam and vsb writing in 12 hrs for 193i," in *Optical Microlithography XXXIII*, vol. 11327. SPIE, 2020, pp. 145–158.
[20] T. Ajayi and D. Blaauw, "Openroad: Toward a self-driving, open-source digital layout implementation tool chain," in *Proceedings of Government Microcircuit Applications and Critical Technology Conference*, 2019.
[21] "FreePDK45," https://eda.ncsu.edu/freepdk/freepdk45/.
[22] S. Banerjee, Z. Li, and S. R. Nassif, "ICCAD-2013 CAD contest in mask optimization and benchmark suite," in *Proc. ICCAD*, 2013, pp. 271–274.