Neural-ILT: Migrating ILT to Neural Networks for Mask Printability and Complexity Co-optimization

Bentian Jiang\textsuperscript{1}, Lixin Liu\textsuperscript{1}, Yuzhe Ma\textsuperscript{1}, Hang Zhang\textsuperscript{2}, Bei Yu\textsuperscript{1} and Evangeline F.Y. Young\textsuperscript{1}

\textsuperscript{1} CSE Dept., The Chinese University of Hong Kong
\textsuperscript{2}ECE Dept., Cornell University
Bentian Jiang is currently pursuing a Ph.D. degree with the Dept. of Computer Science and Engineering, The Chinese University of Hong Kong, under the supervision of Prof. Evangeline F.Y. Young.

He is a recipient of several prizes in renowned EDA contests including the CAD Contests at ICCAD 2018 and ISPD 2018, 2019, 2020.

His research interests include
- Design for manufacturability
- Physical design
Outline

- Introduction and Background
- Neural-ILT Algorithm
- Result Visualization and Discussion
Outline

- Introduction and Background

- Neural-ILT Algorithm

- Result Visualization and Discussion
Background

Lithography

- Use light to transfer a geometric pattern from a photomask to a light-sensitive photoresist on the wafer
- Mismatch between lithography system and device feature sizes

Optical proximity correction (OPC)

- OPC compensates the printing errors by modifying the mask layouts
- Compact lithography simulation model (designed to learn the printing effects) can guide the model-based OPC processes

Inverse Lithography Technology (ILT)

- Forward lithography simulation can mimic the mask printing effects on wafer
  - Given the desired target pattern \( Z_t \), optimized mask \( M \)
  - Forward Lithography simulation produce the corresponding wafer image
    \[
    Z = f(M; P_{\text{nom}})
    \]

- ILT correction tries to find the optimum mask \( M_{\text{opt}} \)
  \[
  M_{\text{opt}} = f^{-1}(Z_t; P_{\text{nom}})
  \]

- Features
  - **Ill-posed**: no explicit closed-form solution for \( f^{-1} \cdot ; P_{\text{nom}} \)
  - **Numerical**: gradient descent to update the on-mask pixels iteratively
  - Pros: best possible overall process window [1] [2] for 193i layers and EUV
  - Cons: drastically computational overhead, unmanageable mask writing time
Motivations

- Tremendous demands
  - Quality: best possible process window obtainable for 193i and EUV layers [1] [2]
  - Manufacturability: unmanageable mask writing times of ideal ILT curvilinear shapes affect high-volume yields
  - Affordability: the still increasing computational overhead

- Goals
  - A purely learning-based end-to-end ILT solution
    - The satisfactory mask printing shapes
    - Breakthrough reduction on computational overhead
    - Significant improvement on mask shape complexity
    - ...
  - A learning-scheme with performance guarantee
Outline

- Introduction and Background
- Neural-ILT Algorithm
- Result Visualization and Discussion
Why Neural Network – Analogy

- What kind of container is needed for end-to-end ILT correction process
  - Layout image in, mask image out
  - Iterative process
  - Update an “object” (mask here) iteratively by gradient descent

- Does it sound like the **training procedure** of an auto-encoder network?
  - Encoder + decoder → Image in, image out
  - Iteratively update neurons of each layer by gradient descent

[Schema of a basic Autoencoder](https://commons.wikimedia.org/w/index.php?curid=80177333)

By Michela Massi - Own work, CC BY-SA 4.0.
Starting from Scratch

- Let us start Neural-ILT with a basic image-to-image translation task.

- Given the sets of
  - Input target layouts $\mathcal{Z}_t = \{\mathcal{Z}_{t,1}, \mathcal{Z}_{t,2}, \mathcal{Z}_{t,3}, \ldots, \mathcal{Z}_{t,n}\}$
  - Corresponding ILT synthesized mask set $\mathcal{M}^* = \{\mathcal{M}_1^*, \mathcal{M}_2^*, \mathcal{M}_3^*, \ldots, \mathcal{M}_n^*\}$

- The training procedure (supervised) of the U-Net is to minimize the objective:

$$\hat{w} = \arg\min_w \lambda \| \phi(\mathcal{Z}_t; w) - \mathcal{M}^* \|^2_2$$
Untrustworthy Quality of Prediction

- Big trouble – Untrustworthy predict quality

(a) Target layouts.
Wafer images generated by:
(b) Target layouts
(c) UNet direct prediction
(d) ILT synthesized masks

- Exists inevitable prediction loss which is not acceptable

- On-neural-network ILT correction is needed to ensure performance
  - Our solution: cast ILT as an **unsupervised** neural-network training procedure
Overview of Neural-ILT

- 3 sub-units:
  - A pre-trained UNet for performing layout-to-mask translation
  - An ILT correction layer for minimizing inverse lithography loss
  - A mask complexity refinement layer for removing redundant complex features

- Core engine:
  - CUDA-based lithography simulator (a partially coherent imaging model)
Challenges on Runtime Bottleneck

- Main computational overhead of ILT correction lies in mask litho-simulation

- Multiple rounds of litho-simulation (per layout, per iteration) are indispensable for guiding the ILT correction

- First critical challenge is to integrate a fast-enough lithography simulator into our Neural-ILT framework
GPU-based Litho-Simulator

- Partially coherent imaging system for lithography model \( f(\mathbf{M}; \mathbf{P}_{\text{nom}}) \)
  - Given the mask \( \mathbf{M} \), litho-sim model parameters \( \omega_k, h_k \), wafer image \( \mathbf{Z} \) can be calculated as
    \[
    I(x, y) = \sum_{k=1}^{N^2} \omega_k |\mathbf{M}(x, y) \otimes h_k(x, y)|^2
    \]
    \[
    Z(x, y) = \begin{cases} 
    1, & \text{if } I(x, y) \geq I_{th} \\
    0, & \text{if } I(x, y) < I_{th} 
    \end{cases}
    \]

- CUDA: perfect for parallelization + demands of AI toolkits integration
  - 96% reduction in litho-simulation time
  - 97% reduction in PVBand calculation time
  - Compatible with popular toolkits: PyTorch, TensorFLow, etc…
ILT Correction Layer

- ILT correction is essentially minimizing the images difference by gradient descent

\[
L_{ilt} = \sum_{x=1}^{N} \sum_{y=1}^{N} (Z(x, y) - Z_t(x, y))^\gamma
\]

- Gradient of \( L_{ilt} \) with respect to mask \( \overline{M} \) (\( M = \text{sigmoid}(\overline{M}) \)) can be derived as

\[
\frac{\partial L_{ilt}}{\partial \overline{M}} = y \times (Z - Z_t)^{\gamma-1} \odot \frac{\partial Z}{\partial \overline{M}} \odot \frac{\partial M}{\partial \overline{M}}
\]

\[
= y \theta_M \theta_Z \times \{H^{\text{flip}} \otimes [(Z - Z_t)^{\gamma-1} \odot Z \odot (1 - Z) \odot (M \otimes H^*)] + (H^{\text{flip}})^* \otimes [(Z - Z_t)^{\gamma-1} \odot Z \odot (1 - Z) \odot (M \otimes H)] \}
\]

\[
\odot M \odot (1 - M)
\]

- where \( Z_t \) is target pattern, \( Z \) is wafer image, \( M \) is mask, \( \omega_k, h_k \) are litho-sim model parameters
ILT Correction Layer

- ILT Correction Layer Implementation
  - Forward to calculate the ilt loss with respect to network prediction and target layout
  - Backward to calculate the gradient mask to update the UNet neurons

- Extremely fast with our GPU-based lithography simulator

- Directly used as a successor layer of other neural networks (expressed in PyTorch)
Complexity Refinement Layer

- ILT synthesized masks
  - Non-rectangular complex shapes
  - Not manufacturing-friendly

- Complex features
  - Isolated curvilinear stains
  - Edge glitches
  - Redundant contours

- Goals
  - Eliminate the redundant/complex features
  - Maintain competitive mask printability
Complexity Refinement Layer

- Complex features are distributed around/on the original patterns

- Observe that, those features
  - Help to improve printability under nominal process condition
  - Not printed under min ($P_{\text{min}}$) / nominal ($P_{\text{nom}}$) process conditions
  - But usually printed under max process condition ($P_{\text{max}}$)

- Cause area variations between
  - $Z_{\text{in}} = f(M; P_{\text{min}})$ and $Z_{\text{out}} = f(M; P_{\text{max}})$
  - Loss function: $L_{\text{cplx}} = \|Z_{\text{in}} - Z_{\text{out}}\|_2^2$.

- Gradient: $\frac{\partial L_{\text{cplx}}}{\partial M} = 2 \times (Z_{\text{in}} - Z_{\text{out}}) \odot (Z_{\text{in}}' - Z_{\text{out}}')$. 
Neural-ILT

- 3 sub-units:
  - A pre-trained UNet for performing layout-to-mask translation
  - An ILT correction layer for minimizing lithography loss
  - A mask complexity refinement layer for removing redundant complex features

- The on-neural-network ILT correction is essentially an *unsupervised training procedure* of Neural-ILT with following objective

\[
\hat{w} = \arg\min_w \alpha \left\{ L_{ilt} \right\} + \beta \left\{ L_{cplx} \right\}
\]

\[
L_{ilt} = \| f(\phi(Z_t; w); P_{\text{nom}}) - Z_t \|_\gamma^2
\]

\[
L_{cplx} = \| f(\phi(Z_t; w); P_{\text{min}}) - f(\phi(Z_t; w); P_{\text{max}}) \|_2^2
\]
All in One Network

- **End-to-end** ILT correction with purely learning-based techniques
- Directly generate the masks after ILT without any additional rigorous refinement on the network output
Retrain Backbone with Domain Knowledge

- Original ILT synthesized training dataset usually consist of numerous complex features
  - We use a Neural-ILT to purify the original training instances

- Use the refined dataset to re-train the UNet with the cycle loss $L_{cycle}$

\[
L_{cycle} = \| \phi(\mathcal{Z}_t; \mathbf{w}) - \mathcal{M}^* \|^2_2 + \eta \| f(\phi(\mathcal{Z}_t; \mathbf{w}); \mathbf{P}_{nom}) - \mathcal{Z}_t \|^2_2
\]

- Domain knowledge of the partially coherent imaging model is introduced into the network training
- ILT is ill-posed, term with domain knowledge serves as a regularization term
- Guide the re-trained network $\phi(\cdot; \mathbf{w})$ gradually converged along a domain-specified direction
- Obtain better initial solution and hence achieve faster convergence
Outline

▪ Introduction and Background

▪ Neural-ILT Algorithm

▪ Result Visualization and Discussion

- On ICCAD 2013 benchmarks
- 70x, 30x TAT speedup
- 12.3%, 3.4% squared L2 error reduction
- 67%, 21% mask fracturing shot count reduction

<table>
<thead>
<tr>
<th>Benchmarks</th>
<th>ILT</th>
<th>PGAN-OPC</th>
<th>Neural-ILT</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Area (nm²)</td>
<td>TAT (s)</td>
<td>L₂ (nm²)</td>
</tr>
<tr>
<td>case1</td>
<td>215344</td>
<td>1280</td>
<td>49893</td>
</tr>
<tr>
<td>case2</td>
<td>169280</td>
<td>381</td>
<td>50369</td>
</tr>
<tr>
<td>case3</td>
<td>213504</td>
<td>1123</td>
<td>81007</td>
</tr>
<tr>
<td>case4</td>
<td>82560</td>
<td>1271</td>
<td>20444</td>
</tr>
<tr>
<td>case5</td>
<td>281958</td>
<td>1120</td>
<td>44656</td>
</tr>
<tr>
<td>case6</td>
<td>286234</td>
<td>391</td>
<td>57375</td>
</tr>
<tr>
<td>case7</td>
<td>292149</td>
<td>406</td>
<td>37221</td>
</tr>
<tr>
<td>case8</td>
<td>128544</td>
<td>388</td>
<td>19782</td>
</tr>
<tr>
<td>case9</td>
<td>317581</td>
<td>1138</td>
<td>55399</td>
</tr>
<tr>
<td>case10</td>
<td>102400</td>
<td>387</td>
<td>24381</td>
</tr>
<tr>
<td>Average</td>
<td>-</td>
<td>788.5</td>
<td>44012.7</td>
</tr>
<tr>
<td>Ratio</td>
<td>-</td>
<td>1.000</td>
<td>1.000</td>
</tr>
</tbody>
</table>
Results

(a) ILT, (b) PGAN-OPC, (c) Neural-ILT

(1) ILT output mask, use 2045 shots to accurately replicate the mask

(2) Neural-ILT output mask, use 653 shots to accurately replicate the mask
Animation: Neural-ILT vs. Conventional ILT

Learning rate (stepsize)

- Neural-ILT is decreasing from $1e^{-3}$
- Convectional ILT is decreasing from 1.0

Neural-ILT correction process
Runtime = 13.57 secs

Conventional ILT correction process
Runtime = 1280 secs
The initial solution of Neural-ILT has much better printability (smaller image errors)
May lead to faster and better convergence
Why Neural Network – Empirical Observation

- GPU-ILT v.s. Neural-ILT, Neural-ILT enjoys
  - Higher searching efficiency: less ILT iterations (i.e., 100 vs. 40)
  - Smooth and fine-grained search: much smaller learning rate (i.e., 1.0 vs. 0.001)
  - Larger searching space: better overall quality (i.e, 9% better printability, 51% less shots counts)

- Reserved inverse lithography function
  - Original ILT loses every internal steps except the final $M_{\text{opt}}$
  - Converged Neural-ILT is indeed an (approximated) inverse lithography function $f^{-1}(\cdot ; \cdot)$ for the given target layout
End
Reference


