Xplace: An Extremely Fast and Extensible Placement Framework

Lixin Liu, Bangqi Fu, Shiju Lin, Jinwei Liu, Evangeline F.Y. Young, *Fellow, IEEE*, and Martin D.F. Wong, *Fellow, IEEE*

***Abstract*—Placement serves as a fundamental step in VLSI physical design. Recently, GPU-based placer DREAMPlace [1] demonstrated its superiority over CPU-based placers. In this work, we develop an extremely fast GPU-accelerated placer Xplace which considers factors at operator-level optimization. Xplace achieves around 2x speedup with better solution quality compared to DREAMPlace. We also plug a novel Fourier neural network into Xplace as an extension. Besides, we enable Xplace to handle the detailed-routability-driven placement problem and demonstrate its superiority in terms of quality and performance. We believe this work not only proposes an extremely fast and extensible placement framework but also illustrates a possibility of incorporating a neural network component into a GPU-accelerated analytical placer.**

***Index Terms*—placement, GPU acceleration, physical design, neural network, routability optimization**

I. Introduction

PLACEMENT serves as a fundamental step in VLSI physical design due to the strong correlation between the placement quality and the circuit's PPA (power, performance and area). Meanwhile, modern circuits contain millions of standard cells, which highly increases the computational complexity of the placement problem and brings huge challenges to the leading-edge global placers.

To tackle the aforementioned problem, various kinds of CPU-based analytical global placers have been proposed over the past decades to optimize the placement half-perimeter wirelength (HPWL). Quadratic placers represent wirelength with a quadratic function and mitigate cell density by cell inflation [2], [3], rough legalization [4], [5], force-directed method [6], [7], cell shifting [8], etc. Although quadratic placers show fast run time to converge, their solution qualities are limited by the low modeling order of the wirelength. Non-linear placers approximate wirelength by a smooth version of the HPWL metric and model cell density by bell-shaped function [9]–[12], Poisson's equation [13]–[17], etc. Different from quadratic placers, non-linear placers produce higher solution quality while the running time overhead is huge. In this work, we mainly consider the non-linear electrostatics-based placement algorithms [15] due to their superior placement solution quality.

With the rapid development of GPU's computational power, GPU acceleration becomes an important direction to pursue to handle large scale problem with parallelism. In global placement, the work [18] accelerated multi-level analytical placer mPL6 [13] with GPU by parallelizing the computation of the wirelength function as well as the spreading force and achieved around 15× speedup. The work [19] explored the idea of utilizing sparse matrix multiplications to compute wirelength and adopting a flattening technique for area computation. However, the maximum wirelength degradation in [18] is larger than 5% and the work [19] does not report their solution quality in details.

Recently, DREAMPlace [1], [20], [21] implemented the approach of ePlace [15] on GPU by casting the placement problem as a neural network training problem and demonstrated the superiority of GPU-accelerated global placers. Compared to [17], DREAMPlace not only produced more than 40x runtime speedup on average for large benchmarks but also provided an open-source analytical placement framework for researchers to further develop. However, it focused on accelerating the wirelength and density operators on GPU while lacking a more general operator-level optimization.

Besides HPWL, routability is another important metric to measure the placer's effectiveness. Previous routability-driven placers [1], [2], [12], [22] perform cell inflation and congestion removal to optimize the routability. However, [1], [22], and [2] only verified their solution by a global router and lacked detailed-routability evaluation. The work [12] handled detailed-routability by a customized detailed placer while invoking a computationally intensive global router during global placement, which largely affected its runtime efficiency.

In this work, we develop Xplace, an efficient yet extensible GPU-accelerated placement framework built on top of PyTorch [23], to consider factors at operator-level optimization. Xplace not only achieves better performance and quality than DREAMPlace but also shows high extensibility to incorporate neural network and routability optimization into analytical placer. The source code of Xplace is released on GitHub[1]. Our key contributions are summarized as follows.

- With operator combination, operator extraction, operator reduction and operator skipping, Xplace achieves around 3x speedup per GP iteration compared to the state-of-the-art global placer DREAMPlace. A placement-stage-aware parameter scheduling technique is also proposed to improve the solution quality. Experimental results

This research was partially supported by ACCESS - AI Chip Center for Emerging Smart Systems, sponsored by InnoHK funding, Hong Kong SAR.

Lixin Liu, Bangqi Fu, Shiju Lin, Jinwei Liu, and Evangeline F.Y. Young are with the Department of Computer Science and Engineering, The Chinese University of Hong Kong (e-mail: lxliu@cse.cuhk.edu.hk).

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

[1] https://github.com/cuhk-eda/Xplace

show that Xplace achieves around 2x speedup with better solution quality compared to DREAMPlace.

- We plug into Xplace a novel Fourier neural network as an extension. The neural network serves as a global guidance for placement. Experimental results not only illustrate a possibility of adopting neural guidance in analytical global placement but also demonstrate Xplace's extensibility in integrating with neural networks.
- We extend Xplace to a fast detailed-routability-driven placer, named Xplace-Route. Equipped with detailed-routability-aware techniques and a GPU-accelerated routing engine, Xplace-Route effectively improves detailed routability and reduces the number of violations. Experimental results on ISPD 2015 contest benchmarks show that Xplace-Route achieves significant quality improvement and runtime speedup.

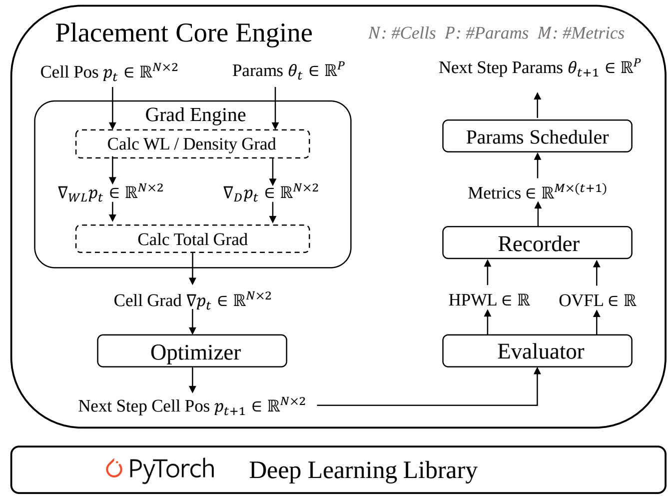


Fig. 1. Overview of Xplace

II. Preliminaries

Given a placement circuit $G = (V, E)$, V represents the set of cells and E denotes the set of nets. Let $p = \{(x_1, y_1), ..., (x_N, y_N)\} \in \mathbb{R}^{N\times 2}$ denote the 2D positions of the cells, where N is the number of cells. The placement region R is uniformly split into an $M_r \times M_c$ grid B. The objective of placement is to minimize the total HPWL of all the nets while satisfying the cell density constraint, which is formulated as,

$$\min_p HPWL(p) = \min_p \sum_{e\in E} HPWL_e(p) \tag{1a}$$

$$\text{s.t.} \quad D_b \le D_t, \forall b \in B \tag{1b}$$

where D_b and D_t denote bin b's cell density and the benchmark-given target density respectively. The HPWL of net e is defined as follows:

$$HPWL_e(p) = (\max_{i\in e} x_i - \min_{i\in e} x_i) + (\max_{i\in e} y_i - \min_{i\in e} y_i). \tag{2}$$

Because the HPWL in Equation (2) is not differentiable, analytical placement reformulates the objective with a smooth approximation of HPWL equipped with a cell density penalty to relax the cell density constraint:

$$\min_p \sum_{e\in E} WL_e(p) + \lambda D(p) \tag{3}$$

where the wirelength $WL_e(p) = WL_e(x) + WL_e(y)$ is modeled as the weighted-average (WA) [24] wirelength with a coefficient γ,

$$WL_e(x) = \frac{\sum_{i\in e} x_i e^{x_i/\gamma}}{\sum_{i\in e} e^{x_i/\gamma}} - \frac{\sum_{i\in e} x_i e^{-x_i/\gamma}}{\sum_{i\in e} e^{-x_i/\gamma}} \tag{4}$$

and similarly for $WL_e(y)$. A smaller γ leads to more accurate approximation of HPWL. The cell density penalty $D(p)$ is formulated in ePlace [15], which treats the cell density problem as analogous to an electrostatic system. This method achieves uniform cell distribution through the application of electric force. The parameter λ controls the weight of the cell density penalty $D(p)$. A typical placement flow starts with a small λ and gradually increase it to remove cell overlaps.

In ePlace [15] on which Xplace is based, each cell i is modeled as a charge, and the cell density is modeled as an electrostatic system denoted as:

$$\begin{cases} \nabla \cdot \nabla\psi(x,y) = -\rho(x,y), \\ \hat{\mathbf{n}} \cdot \nabla\psi(x,y) = \mathbf{0}, (x,y) \in \partial R, \\ \iint_R \rho(x,y) = \iint_R \psi(x,y) = 0, \end{cases} \tag{5}$$

where ∂R is the boundary of the placement region, $\rho(x,y)$ is the electron density map, $\psi(x,y)$ is the electric potential distribution, and $\xi(x,y) = -\nabla\psi(x,y)$ is the electric field distribution. The numerical solution of Poisson's equation in Equation (5) is derived by discrete cosine transformation (DCT) in [15].

III. Overview

In this section, we will discuss the design of Xplace, a fast and extensible GPU-accelerated placer. Our Xplace framework is shown in Fig. 1. Xplace is built on top of PyTorch and contains a placement core engine. Inside the core engine, the gradient engine takes cell position and placement parameters as input to compute the cell gradient. Next, the optimizer utilizes the computed gradient to update the cell position. The evaluator evaluates the placement solution, and the recorder records the placement metrics like HPWL and overflow. Finally, the scheduler decides how to modify the parameters and whether to stop the global placement. It is worth noting that all these parts are designed as independent modules in Xplace so that one can easily extend Xplace by applying new scheduling techniques, new gradient functions, new placement metrics and so on.

Section IV will discuss several important technical details that enable Xplace running very efficiently on GPU. We propose operator-level optimization techniques to achieve effective parallelization. Section V will discuss a placement-stage-aware parameters scheduling to improve the solution quality. Section VI will extend Xplace with a Fourier neural operator to show its high extensibility. Section VII will describe a routability-driven placement flow to optimize the detailed routability.

IV. Operator-Level Optimization

A. Wirelength Operator Combination

Weighted average (WA) [24] wirelength is used as the wirelength objective in many analytical global placers. In Xplace, we adopt the WA wirelength in Equation (4) as our wirelength objective and update the cell position based on the guidance of the WA gradient. To avoid numerical overflow in Equation (4), a mathematically equivalent but numerically stable version is given in Equation (6), which requires the minimum and maximum position among all pins in a net.

$$WL_e(x) = \frac{\sum_{i\in e} x_i e^{\frac{x_i - \max_{j\in e} x_j}{\gamma}}}{\sum_{i\in e} e^{\frac{x_i - \max_{j\in e} x_j}{\gamma}}} - \frac{\sum_{i\in e} x_i e^{\frac{\min_{j\in e} x_j - x_i}{\gamma}}}{\sum_{i\in e} e^{\frac{\min_{j\in e} x_j - x_i}{\gamma}}} \tag{6}$$

A wirelength objective-and-gradient merging method, proposed in [1], computes both the WA wirelength and the WA gradient within a single GPU thread to mitigate memory bounded problems.

Since both the HPWL function in Equation (2) and the stable WA wirelength function in Equation (6) need the minimum and maximum cell positions in a net, we further modify this merging method by combining the three operators with heavy wirelength-related workload, WA wirelength, WA gradient and HPWL, into one operator to avoid redundant computation of the minimum and maximum function. The proposed operator combination technique can significantly reduce the total GPU execution time.

B. Density Operator Extraction

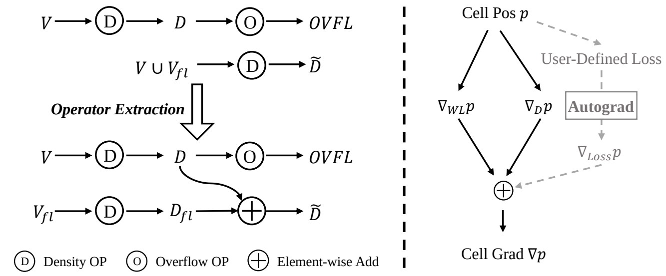
Density objective is one of the most computationally intensive operators in global placement. Similar to [1] and [19], we implemented a GPU-accelerated area accumulation operator to compute the cell density map and apply PyTorch built-in rfft2/irfft2 operators to derive the numerical gradient of the electrostatic system in Equation (5). We also implemented a GPU-accelerated version of the overflow ratio operator $OVFL$, whose CPU version is applied in NTUplace3 [10] and ePlace [16], to measure the evenness of cell distribution and guide the parameter update. The overflow ratio is described as,

$$OVFL = \frac{\sum_{b\in B} \max(D_b - D_t, 0)A_b}{\sum_{i\in V_{mov}} A_i} \tag{7}$$

where A_b and A_i denote the area for bin b and cell i; D_b is bin b's cell density; D_t is the target density, and V_{mov} is the set of movable cells. Concretely, each bin b's cell density D_b in the cell density map $D \in \mathbb{R}^{M_r\times M_c}$ is defined as,

$$D_b = \frac{\sum_{i\in V} A_i \cap A_b}{A_b},\ \forall b \in B \tag{8}$$

where $A_i \cap A_b$ defines the overlap area between cell i and bin b. Similar to [16] and [25], we insert filler cells inside the electrostatic system to handle whitespace and prevent the density objective to overly spread the cells. The inserted filler density map $D_{fl} \in \mathbb{R}^{M_r\times M_c}$ is given as,



(a) Operator Extraction (b) Cell Gradient Computation

Fig. 2. Illustration for the operator extraction technique and the cell gradient computation scheme.

$$D_{fl,b} = \frac{\sum_{i\in V_{fl}} A_i \cap A_b}{A_b},\ \forall b \in B, \tag{9}$$

where V_{fl} denotes the set of filler cells. Therefore the total density map $\tilde{D} \in \mathbb{R}^{M_r\times M_c}$ used for solving the electrostatic system is formulated as follows,

$$\tilde{D}_b = \frac{\sum_{i\in V\cup V_{fl}} A_i \cap A_b}{A_b} \tag{10a}$$

$$= \frac{\sum_{i\in V} A_i \cap A_b}{A_b} + \frac{\sum_{i\in V_{fl}} A_i \cap A_b}{A_b} \tag{10b}$$

$$= D_b + D_{fl,b},\ \forall b \in B \tag{10c}$$

Then, the matrix form of Equation (10) is formulated as

$$\tilde{D} = D + D_{fl} \tag{11}$$

We observe that both Equation (8) and Equation (11) contain the computation of cell density map D. Due to the heavy load of the density map operator, performing a common sub-operator extraction in Equation (11) will naturally boost the performance. As shown in Fig. 2(a), we first compute the cell density map D and the filler density map D_{fl} separately. We then adopt the element-wise add operator to compute the total density map $\tilde{D}$ and apply the overflow operator to calculate $OVFL$. Note that overflow ratio computation is needed for updating parameters in each GP iteration. The proposed sub-operator extraction technique will reduce the total computation time of the cell density map D and achieve a visible improvement in the total GPU execution time.

C. Operator Reduction

PyTorch [23] is a well-known deep learning library that provides a lot of built-in differentiable operators (e.g. element-wise addition, matrix multiplication, convolution, etc.). In forward propagation, users can apply the provided/user-defined operators to construct a neural network or a gradient-based optimization. In backward propagation, an automatic differentiation (autograd) engine is invoked to compute derivatives automatically. Although PyTorch makes development convenient, there are technical details that need to be carefully considered when building a global placer using PyTorch.

In PyTorch, the execution of each operator will perform a kernel launching step on CPU before executing the core CUDA kernel on GPU. Not only that the forward propagation will execute operators but also the backward propagation, driven by the autograd engine, need to run gradient operators. However, the kernel launching overheads of these operators may even be much larger than their GPU execution overheads when their computation workloads are small. An alternative method to distinguish between CPU-launching-dominated and GPU-execution-dominated kernels is through profiling. In general, except for the heavily loaded operators (e.g. wirelength and density computations) that are related to the netlist size and the die area, the kernel launching overheads of most other placement operators are much larger than their GPU execution overheads. In this case, the more operators being executed, the larger the total kernel launching overhead there will be. If the total kernel launching overhead dominates the GPU execution time, the speedup will be limited.

To this end, we propose operator reduction to reduce the number of operators to mitigate the aforementioned issue. The main idea of operator reduction is to avoid invoking the heavy autograd engine. Since the number of forward operators are almost the same as that in the backward, invoking the heavy autograd engine will almost double the number of operators and bring large kernel launching overhead on CPU. To resolve this problem, we directly derive the numerical solutions of the wirelength gradient and the density gradient without invoking the autograd engine and assign a weighted accumulated gradient to each cell. This step can reduce the total kernel launching time and boost the performance significantly. It is worth noting that avoiding invoking the autograd engine will not affect our framework's extensibility since PyTorch also supports invoking the autograd engine for user-defined loss function and accumulating the separately computed numerical gradient with the backward gradient of the user defined loss function as illustrated in Fig. 2(b).

Besides operator reduction, the PyTorch in-place operators (e.g. torch.add_, torch.mul_, etc.), which directly manipulate the tensor memory without memory copying, are used as much as possible. For instance, there's no need to allocate new GPU memory to store certain intermediate variables. Instead, we can simply manipulate the GPU memory that was allocated previously to avoid redundant copying.

Finally, as frequent synchronization will interrupt the GPU pipeline and slow down the total run time, we reorder the operators that need synchronization to the end of the execution queue in each GP iteration so that the negative effects of synchronization can be alleviated.

D. Operator Skipping

It is observed that the ratio between density gradient and wirelength gradient, given as follows,

$$r = \frac{\lambda |\nabla D_{x,y}|}{|\nabla WL_{x,y}|} \tag{12}$$

is ultra-small in the early placement stage. Based on the observation, we propose an early-stage operator skipping technique to further boost the runtime performance. When $(r < 0.01) \wedge (iteration < 100)$, the density gradient operator will only be executed once per 20 iterations.

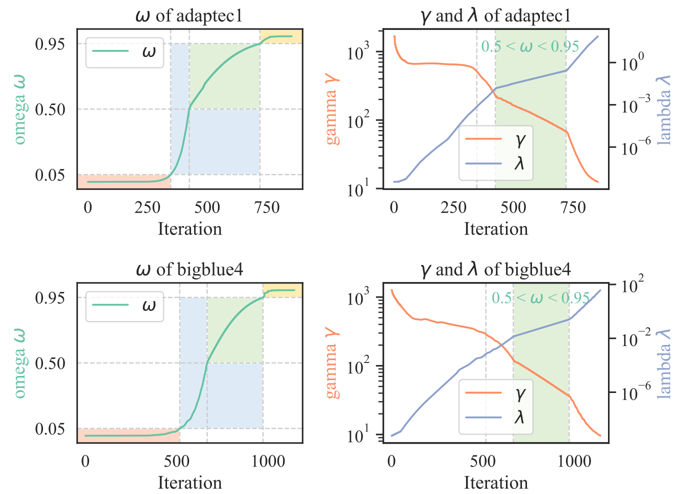


Fig. 3. Illustrations of the precondition weighted ratio ω and its influence on ISPD 2005 `adaptec1` and `bigblue4`. The effects of slowing update on wirelength coefficient γ and density weight λ are shown in the green area within the figures on the right-hand side.

E. Determinism

Similar to DREAMPlace [1], we fix the non-determinism in the DAC version Xplace [26] by converting the floating point numbers to fixed point to avoid non-deterministic floating-point atomic-add operations. Technically, the non-determinism in placement comes from the computation of the density map D. Because $\sum_{x,y} D_{x,y} \leq M_r \times M_c$ and $D_{x,y} \geq 0 \ \forall x, y$ [15], where M_r and M_c is the number of rows and columns in D, we have $D_{x,y} \leq M_r \times M_c \ \forall x, y$. Following DREAMPlace [1], we apply a scaling factor $2^{64-\max(\lceil \log_2 M_r \times M_c \rceil + 1, K_s)}$ to scale up the density map computation and convert it from float32 to uint64. Note that K_s can be a constant number and it only influences the precision. We use $K_s = 32$. With the fixed-point conversion, the non-determinism can be avoided.

To store the converted uint64 GPU data, extra memory allocation is needed. We observe that the allocated GPU memory size remains constant at $M_r \times M_c \times$ sizeof(uint64) in each GP iteration. Dynamically allocating memory in this context will cause frequent CPU-GPU synchronization, interrupting the execution pipeline. Thus, we apply a GPU memory pre-allocation technique to avoid unnecessary synchronization and significantly accelerate the deterministic mode.

V. Stage-Aware Parameters Scheduling

Preconditioning is widely used in global placement [7], [16], [27]. Given in Equation (13), preconditioning is applied to the optimization objective for reducing the condition number and accelerating Nesterov's optimization convergence.

$$\mathbf{H}^{-1} = (\mathbf{H_W} + \lambda \mathbf{H_D})^{-1}, \tag{13a}$$

$$\mathbf{H_W} = \text{diag}(|S_1|, |S_2|, ..., |S_{|V|}|), \tag{13b}$$

$$\mathbf{H_D} = \text{diag}(A_1, A_2, ..., A_{|V|}), \tag{13c}$$

Algorithm 1 Placement-Stage-Aware Parameters Scheduling

```
1: γ ← γ_0                                              ▷ wirelength coefficient
2: λ ← λ_0                                              ▷ density weight
3: while iteration < ITER and NOT Convergence do
4:     if 0.5 < ω < 0.95 and iteration%3 ≠ 0 then
5:         SKIP_UPDATE
6:     else
7:         γ ← γ × coef(overflow)
8:         λ ← λ × μ(Δhpwl)        ▷ γ and λ are derived from [17]
9:     ω ← λ|H_D| / (|H_W| + λ|H_D|)
```

where $S_i = \{e\}_i$ is the set of nets containing cell i, $|S_i|$ is the number of nets connecting cell i, A_i is the area of cell i, and λ is the weight of the density penalty. Recall that the placement objective is formulated as,

$$\min_p WL(p) + \lambda D(p).$$

With the preconditioner, the modified cell gradient is derived as follows,

$$\nabla p = (\mathbf{H_W} + \lambda \mathbf{H_D})^{-1}(\nabla WL_{x,y} + \lambda \nabla D_{x,y}), \tag{14}$$

where ∇p denotes the cells' gradient.

To measure the placement stage, we introduce the precondition weighted ratio $\omega \in (0, 1)$,

$$\omega(\lambda) = \lambda |\mathbf{H_D}| |\mathbf{H_W} + \lambda \mathbf{H_D}|^{-1} \tag{15a}$$

$$= \frac{\lambda \sum_{i \in V} A_i}{\sum_{i \in V} |S_i| + \lambda \sum_{i \in V} A_i}, \tag{15b}$$

where $|\cdot|$ is the L_1-norm operator. Note that ω is only determined by the variable λ (A_i and S_i are only related to the design technology), where λ is gradually updated during placement to enhance the importance of the density objective. Compared to $\frac{\lambda}{1+\lambda}$, the precondition weighted ratio $\omega(\lambda)$ is more relevant to the cell gradient formulated in Equation (14) and it further reflects the relative importance of $\lambda \mathbf{H_D}$ in $\mathbf{H}^{-1}$.

Through our investigation, $\omega(\lambda)$ successfully measures the global placement optimization stage. As shown in Fig. 3, ω gradually increases when the placement iterates. When $\omega < 0.05$ (marked in red), the optimization objective is wirelength-dominated, and cells are driven to the position with minimum wirelength. As ω rapidly grows in the intermediate stage ($0.05 < \omega < 0.95$, marked in blue), cells spread over the whole placement region, and the overlap ratio significantly decreases. At the end of the placement ($\omega > 0.95$, marked in yellow), cells are forced to a final position with minimum local penalty at the final stage. Note that the first-order derivatives of $\omega(\lambda)$ in the red and yellow areas are both small, while that in the blue area is relatively larger.

If we slow down the parameter update in the blue stage, the gradient optimizer will search the solution spaces more fine-grained, and the quality of the final solution will become better. However, a slower update of placement parameters would affect the running time so we only adopt the above slowing update technique when $0.5 < \omega < 0.95$ (marked in green in Fig. 3) to exploit the optimization space while saving the runtime. A detailed description of this technique is given by Algorithm 1.

VI. Extending Xplace via Neural Enhancement

In this section, we show Xplace's high extensibility by incorporating a deep neural network into its optimization process. As Xplace has a similar architecture to a normal neural network of PyTorch [23], it is natural and easy to embed a neural network as an extension. In this context, we introduce a neural-plugged-in framework, aimed at exploring the possibility of learning-based frameworks. We also demonstrate Xplace's capability of integrating with a neural network.

Neural networks have shown great potential on mapping between dynamic systems defined by partial differential equations (PDE). Previous works of image-to-image mapping tasks are usually conducted in the spatial domain and are accurate on training datasets. However, when given a new pattern of instance, they do not perform well. Recently, convolution operation in the frequency domain, namely Fourier-Neural-Operator (FNO), is introduced in [28]. Given that FFTs have traditionally played an essential role in solving partial differential equations (PDEs), FNO has demonstrated its power in tackling dynamic systems determined by a family of PDEs. Such FNO is capable of learning the universal solution of a dynamic system, even when provided with limited training data.

In the global placement problem, the function of the electric field Equation (5) modeled by Poisson's equation can be considered as a dynamic system mapping from electron distribution to electric field, in which electron distribution is the 2-D density map of placement and the electric field is the unit moving force on x and y-axis. In light of this analogy, we employ an FNO-like model to learn the process of solving Poisson's equation of placement density.

As illustrated in Fig. 4, our model is a two-path convolution network consisting of a spatial-domain path (blue) to extract the explicit information of a specific feature map and a frequency-domain path (orange) to generalize the global information of a continuous dynamic system. In order to transfer the model to multi-resolution, the input density map D is concatenated with mesh-grid index $M_x(x, y) = \frac{x}{X}$ and $M_y(x, y) = \frac{y}{Y}$, where X and Y are map sizes. The input map $I = \{D; M_x; M_y\}$ is firstly lifted to multi-channel by a fully-connected layer, denoted as $I_m = FC(I)$, and then fed into two paths.

In the Fourier path, the spatial map is transformed into the frequency space after Fast-Fourier-Transform (FFT) function $\mathcal{F}$. A low-pass-filter (LPF) L preserves a number of lower frequency components and then a linear transformation $\mathcal{W}$ is applied to the filtered map. After applying an Inverse-Fast-Fourier-Transform (IFFT) function $\mathcal{F}^{-1}$, the frequency map is transformed back to the spatial domain,

$$Freq_{layer}(I_m) = \mathcal{F}^{-1}\Big(\mathcal{W}^T \cdot L(\mathcal{F}(I_m))\Big) \tag{16}$$

In the spatial path, a simple pixel-wise convolution layer is operated on the feature map. Maps from two paths are added followed with a nonlinear activation function $GELU$. The

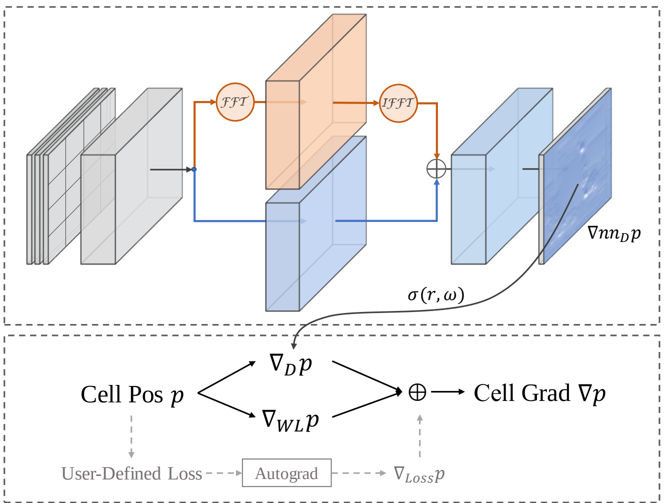


Fig. 4. Neural-network plugged-in placement extension.

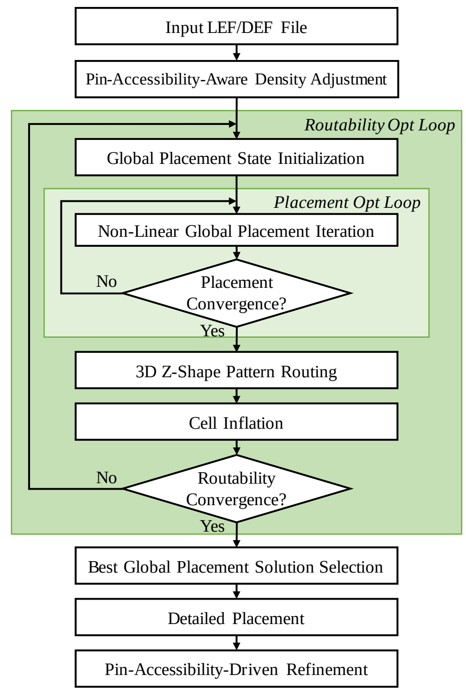


Fig. 5. Routability Optimization Flow of Xplace-Route.

above process of a FNO is described as

$$\mathcal{O}(I_m) = GELU\Big(Conv_{2D}(I_m) + Freq_{layer}(I_m)\Big) \tag{17}$$

We then get the output after a down-sampling fully-connected layer $FC^{-1}\Big(\mathcal{O}(I_m)\Big)$ and a relative L_2 loss function is used for back-propagation:

$$L_2(\mathbf{x_i}, f(\mathbf{x_i};\theta)) = ||f(\mathbf{x_i},\theta) - \mathbf{y_i}||_2/||\mathbf{y_i}||_2 \tag{18}$$

where $\mathbf{x}_i, \mathbf{y}_i, f$ and θ are the i-th input, label, network model and network parameters. The label and prediction are normalized for evaluation.

During the training process, the model gradually acquires the knowledge of solving the density problem in Equation (5). As a result, there is no necessity to collect actual ground-truth training data from real placement benchmarks. Instead, we can generate randomly distributed density maps and compute the numerical solution of the corresponding electric fields, which will be used as labels for training.

Since we do a pixel-wise convolution on the spatial maps, the resolution of the input maps will not affect the convolution results. Moreover, in Fourier space, low-frequency components describe the global information, while high-frequency components describe the explicit local information. That means, a low-resolution image and a high-resolution image will share similar low-frequency components, with differences in high-frequency components only. As we only preserve a certain number of lower-frequency components, our model is resolution-independent. The model can be trained on low-resolution data and extended to high-resolution, which improves the adaptability of the model.

The electric fields on both the x and y directions share the same partial differential function, with only a difference in the direction. Therefore the model can be trained in only one direction and still be workable in the other direction by simply flipping the input feature map, further improving the generalization of this model.

As the model is trained on low-resolution data and only preserves lower-frequency components, the predicted density gradient will have a good global view to spread cells. Thus we insert the nn-predicted density gradient $\nabla_{nn} D_{x,y}$ into the early stage of placement to help push cells around. With the precondition weighted ratio ω defined in Section V and the gradient ratio r defined in Section IV-D, a smooth function $\sigma(r,\omega)$ is used to weighted-sum with the numerical solution of the density gradient, given as follows,

$$\sigma(r,\omega) = \frac{1}{1+e^{-5(r/0.005-0.5)}} - \frac{1}{1+e^{-5(\omega/0.05-0.5)}} \tag{19}$$

$$\nabla^{'} D_{x,y} = (1-\sigma)\nabla D_{x,y} + \sigma \nabla_{nn} D_{x,y} \tag{20}$$

Function σ has a bell shape, and it smoothly integrates the nn-predicted gradient with the numerical solution. When σ increases, $\nabla_{nn} D_{x,y}$ will dominate. When σ drops, $\nabla D_{x,y}$ takes effect for fine-grained placement. Note that directly introducing the external nn-predicted gradient would cause the divergence in Nesterov's optimization but using the above smoothness successfully avoids that and helps the convergence.

VII. Extending Xplace via Detailed-Routability Optimization

Previous routability-driven placers lacked handling detailed-routability optimization or invoked a time-consuming CPU-based global router for routability optimization.

To tackle the aforementioned issue and fully exploit the power of GPU, we extend the deterministic version Xplace to a detailed-routability-driven GPU-accelerated placer, named Xplace-Route, to effectively handle the detailed-routability. A GPU-accelerated routing engine [29] is implemented in

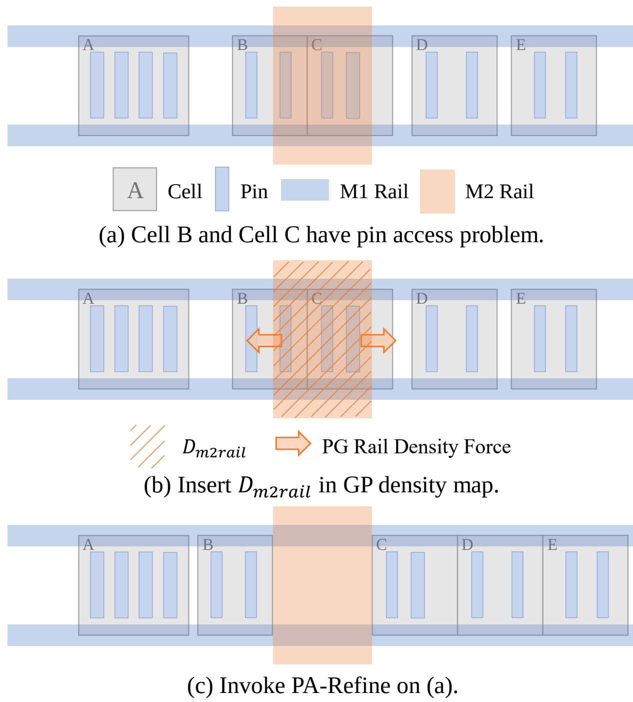


Fig. 6. Pin-Accessibility-Aware Optimization Techniques.

Xplace-Route for routability evaluation and several optimization techniques are adopted to systematically boost the solution's detailed-routability. Experimental results show that Xplace-Route achieves superior detailed-routability and remarkable runtime speedup.

Our proposed routability optimization flow is shown in Fig. 5. Xplace-Route first conducts pin-accessibility-aware density adjustment after parsing the given design. Then Xplace-Route recursively improves the placement routability by a two-level nesting loop until the flow converges. The inner loop aims to optimize the cell position by the non-linear global placement objective given in Equation (3). With the converged placement solution given by the inner loop, Xplace-Route invokes an internal GPU-accelerated pattern router and conducts a cell inflation-based routability optimization loop. Xplace-Route will recursively re-launch the placement engine and further optimize the routability. When the nesting loop converges, Xplace-Route selects the best placement solution according to the historical routing metrics and then performs detailed placement. Finally, a pin-accessibility-driven refinement is adopted to optimize the routability further.

A. Pin-Accessibility-Aware Density Adjustment

Pin-Accessibility is highly related to detailed routability. As shown in Fig. 6(a), the cell B's and C's M1 signal pins are covered by an M2 power and ground (PG) rail, and it is difficult for a routing wire to access these pins without violations because of the limited routing resource on M1. To mitigate the pin-accessibility issue, we insert the PG rail density in global placement as a penalty to move the cells away from the M2 rail as illustrated in Fig. 6(b). Besides, we observe that the area near I/O pin is usually congested, so we increase the I/O pin density to relax the congestion. To this end, the cell density D previously formulated in Equation (8) is modified as follows,

$$D_{m2rail} = \frac{\sum_{i \in V_{m2rail}} A_i \cap A_b}{A_b}, \ \forall b \in B \tag{21a}$$

$$D_{iopin} = \frac{\sum_{i \in V_{iopin}} A_i \cap A_b}{A_b}, \ \forall b \in B \tag{21b}$$

$$D_{route} = D + D_{m2rail} + 3 \times D_{iopin} \tag{21c}$$

where V_{m2rail} and V_{iopin} refer to a set of M2 PG rails and a set of I/O pins respectively. The total density $\tilde{D}$ in Equation (11) used for solving Poisson's Equation is updated correspondingly as follows,

$$\tilde{D} = D_{route} + D_{fl} \tag{22}$$

Note that we can consider M2 rail and I/O pin as fixed macro, thus the D_{m2rail} and D_{iopin} can be pre-computed.

B. Routing Congestion Map

Similar to CUGR [30] and NCTUgr [31], we divide the 3D routing region into a set of global routing cells (G-Cells) $G_R \in \mathbb{R}^{R_r \times R_c \times L}$, where R_r and R_c denote the number of rows and columns in the routing grid graph; L denotes the number of routing layers.

In Xplace-Route, we implement a GPU-accelerated 3D Z-shape pattern routing algorithm discussed in [29] to effectively evaluate the routability of the intermediate placement solution and provide valuable guidance for routability optimization. Reported by the internal router, we obtain the routing demand map $Dmd \in \mathbb{R}^{R_r \times R_c \times L}$ and the routing capacity map $Cap \in \mathbb{R}^{R_r \times R_c \times L}$ of a placement solution, and then we construct a routing congestion map $C \in \mathbb{R}^{R_r \times R_c}$, formulated as follows,

$$C_{x,y} = \max\left(\frac{\sum_{l=1}^{L} Dmd_{l,x,y}}{\sum_{l=1}^{L} Cap_{l,x,y}} - 1, 0\right), \ \forall (x,y) \in \mathbb{R}^{R_r \times R_c} \tag{23}$$

where Dmd is the summation of wire demand and the via demand.

Due to the large runtime overhead of maze routing, we choose only to launch the Z-shape pattern router (PR) to compute the congestion map and guide the cell inflation. Empirically, such a PR-only scheme successfully enhances the placement routability with a relatively small runtime overhead.

C. Cell Inflation

Recent works [1]–[3], [12], [17], [22], [32] demonstrate that inflating cells in the congested area will significantly reduce the routing congestion. In Xplace-Route, we employ a similar cell inflation method from DREAMPlace [1], which utilizes the routing congestion map to compute the cell inflation ratio. The historical cell inflation ratio is given as follows,

$$r_{i,t} = r_{i,t-1} \times \sqrt{\Delta r_{i,t}} \tag{24a}$$

$$\Delta r_{i,t} = \sum_{(x,y) \in \mathbb{R}^{R_r \times R_c}, A_i \bigcap A_{x,y} \neq \varnothing} \frac{A_i \bigcap A_{x,y}}{A_i} IR_{x,y} \tag{24b}$$

where $r_{i,t}$ denotes the inflation ratio of cell i's width and height in the t-th routability optimization iteration; $r_{i,0}$ is initialized as 1; $IR \in \mathbb{R}^{R_r \times R_c}$ is the inflation ratio map, given as follows,

$$IR_{x,y} = \min((C_{x,y}+1)^2, 2),\ \forall(x,y) \in \mathbb{R}^{R_r \times R_c} \tag{25}$$

As formulated in Equation (25), a cell in a more congested area will be much more inflated than one in a non-congested area. If a cell is frequently placed in a congested area, the historical inflation ratio formulated in Equation (24) implies that this cell will be inflated more than once to mitigate the potential routing resource scarcity.

After the calculation of the inflation ratio, Xplace-Route will inflate cells and then re-initialize parameters. Different from the routability-driven DREAMPlace [1], which resets the density weight λ if the overflow ratio is smaller than a certain number, Xplace-Route waits for the global placement convergence and then re-initializes the placement state (including density weight λ, WA coefficient γ, Nesterov optimizer, etc). Since the cells are inflated, the non-linear optimized solution is largely changed. Therefore, re-initialization is necessary for the non-linear optimizer. Empirically, the re-initialization strategy assists the optimizer in searching for a better placement solution.

D. Solution Selection Criteria

Xplace-Route recursively launches the routability optimization to reduce congestion. The routability optimization loop will be terminated if there is insufficient space to inflate the cells or the loop reaches the maximum iteration I_{route} (we set $I_{route} = 5$). After finishing the routability optimization, Xplace-Route selects the placement solution with the smallest routing cost among all the converged placement solutions (i.e. the outputs of the non-linear optimization loop). For a specific placement solution, its routing cost c_{route} is derived from the router-provided wire and via information, given as follows,

$$A_{wire,l} = WireWidth_l \times GCellSize_l \tag{26a}$$

$$c_{wire} = \sum_{l,x,y} \max(WireDmd_{l,x,y} - Cap_{l,x,y}, 0)A_{wire,l} \tag{26b}$$

$$m_{l,x,y} = \begin{cases} 1, & \text{if } WireDmd_{l,x,y} > Cap_{l,x,y}, \\ 0, & \text{otherwise.} \end{cases} \tag{26c}$$

$$c_{via} = \sum_{l,x,y} m_{l,x,y} \times ViaDmd_{l,x,y} \tag{26d}$$

$$c_{route} = c_{wire} + c_{via} \tag{26e}$$

where $A_{wire,l}$ denotes the unit wire area on the l-th routing layer, and c_{wire} refers to the total wire overflow areas. Note that we incorporate $A_{wire,l}$ as a weight within c_{wire} to penalize overflowed G-cells (i.e. G-cells with insufficient routing capacity to accommodate the required wire demand), especially on layers with larger unit wire areas. Besides, for an overflowed G-cell, we take into account its via usage as an additional penalization term, which is formulated as c_{via}. Different from ACE [33], which computes the average of the top $x\%$ congestion, our routing cost metric c_{route} further considers both the unit wire area and the relationship between the routing capacity and demand.

TABLE I
BENCHMARKS STATISTICS

Benchmarks	Design	#cells	#nets	Design	#cells	#nets
ISPD 2005	adaptec1	211k	221k	bigblue1	278k	284k
	adaptec2	255k	266k	bigblue2	558k	577k
	adaptec3	452k	467k	bigblue3	1097k	1123k
	adaptec4	496k	516k	bigblue4	2177k	2230k
ISPD 2015	fft_1	35k	33k	des_perf_1	113k	113k
	fft_2	35k	33k	des_perf_a	108k	115k
	fft_a	34k	32k	des_perf_b	113k	113k
	fft_b	34k	32k	edit_dist_a	127k	134k
	matrix_mult_1	160k	159k	matrix_mult_b	146k	152k
	matrix_mult_2	160k	159k	matrix_mult_c	146k	152k
	matrix_mult_a	154k	154k	pci_bridge32_a	30k	34k
	superblue12	1293k	1293k	pci_bridge32_b	29k	33k
	superblue14	634k	620k	superblue11_a	926k	936k
	superblue19	522k	512k	superblue16_a	680k	697k

E. Pin-Accessibility-Driven Refinement

We integrate with ABCDPlace [34] to perform legalization and detailed placement on the selected global placement solution. However, ABCDPlace lacks detailed-routability-driven refinement. As discussed in Section VII-A, the pin accessibility of the placement solution can be significantly improved by moving cells outside the area covered by M2 rail. Different from [12] and [35], which adopt the pin accessibility optimization during the legalization, we propose a pin-accessibility post-refinement technique, invoked at the end of the detailed placement, to refine the PG rail-related detailed-routability.

Pin-accessibility post-refinement (shortened as PA-refine) works in a row-based manner and post-refines the legal placement solution. If a cell i overlaps with the M2 rail (i.e. the cell i is placed under the M2 rail), PA-refine will search its horizontal neighborhoods for possible movement. If it is possible to shift no more than K cells in the left or right direction to remove the overlap, PA-refine will move these cells sequentially to resolve the pin-accessibility issue. If there is insufficient space for overlap removal within K cells, we won't move them. Empirically, we use $K = 5$ to improve the pin-accessibility while avoiding large displacement. Fig. 6(c) illustrates an example of the result of the above technique.

VIII. EXPERIMENTAL RESULTS

The Xplace framework is developed with PyTorch and CUDA. Unless specified, the experiments are default conducted on a Linux machine with2.90GHz Intel Xeon CPU and a single Nvidia RTX 3090 GPU. Extended from the DAC version [26], we integrate Xplace with the legalization method discussed in DREAMPlace [1] and the detailed placer ABCDPlace [34]. Besides, we implement the deterministic mode without significantly affecting the runtime.

We test our GPU-accelerated placer on the ISPD 2005 contest benchmarks [36] and the ISPD 2015 contest benchmarks [37] with fence-region constraints removed[2]. DREAMPlace [1] is a placement framework accelerated by GPU

[2]This work mainly focuses on developing an extremely fast yet extensible framework. We will address the fence region constraint in our future version.

TABLE II

HPWL($\times 10^6$) AND RUNTIME (SECONDS) RESULTS ON THE ISPD 2005 CONTEST BENCHMARKS [36]. "DM" DENOTES DETERMINISM. THE BEST RESULTS ARE MARKED IN **BOLD**, AND THE SECOND-BEST RESULTS ARE INDICATED IN BROWN.

Design	Xplace			DREAMPlace			XplaceDM			DREAMPlaceDM			Xplace-NN (DM)		
	HPWL	GP/s	PL/s	HPWL	GP/s	PL/s	HPWL	GP/s	PL/s	HPWL	GP/s	PL/s	HPWL	GP/s	PL/s
adaptec1	73.09	**1.47**	13.67	73.20	3.94	19.40	73.08	1.47	**13.57**	73.16	6.18	21.93	**73.08**	2.78	18.36
adaptec2	**81.30**	**1.61**	**15.04**	82.18	4.08	22.76	81.32	2.03	15.58	82.09	6.59	26.02	81.63	3.72	17.19
adaptec3	193.62	**2.48**	**27.09**	**193.26**	4.66	36.71	193.66	2.92	27.87	193.31	7.15	40.42	193.73	4.79	29.58
adaptec4	173.36	**2.85**	**26.17**	174.14	5.14	38.04	173.35	3.39	26.37	173.88	7.99	42.13	**173.30**	5.27	28.72
bigblue1	89.08	**1.47**	**14.45**	89.37	4.28	23.76	89.07	1.65	14.68	89.35	7.43	27.66	**88.97**	2.95	16.51
bigblue2	136.91	**2.45**	38.36	136.92	4.98	48.25	136.91	2.69	**38.20**	136.99	7.31	52.39	**136.42**	4.68	40.18
bigblue3	303.08	**5.53**	**54.15**	304.38	8.24	75.21	303.18	6.91	55.55	304.27	13.75	82.07	**302.14**	9.92	57.93
bigblue4	742.19	**11.80**	**115.52**	744.11	13.63	150.80	742.23	14.26	116.97	744.10	20.26	170.08	**741.02**	18.54	121.21
Mean	224.08	**3.71**	**38.06**	224.70	6.12	51.87	224.10	4.42	38.60	224.64	9.58	57.84	**223.79**	6.58	41.21
Ratio	1.0000	**1.0000**	**1.0000**	1.0028	1.6504	1.3629	1.0001	1.1908	1.0143	1.0025	2.5846	1.5198	**0.9987**	1.7751	1.0829

TABLE III

HPWL($\times 10^6$), TOP5 OVERFLOW, AND RUNTIME (SECONDS) RESULTS ON THE ISPD 2015 CONTEST BENCHMARKS [37]. OVFL-5 STANDS FOR TOP5 OVERFLOW. "DM" DENOTES DETERMINISM. THE BENCHMARKS WITH FENCE REGION CONSTRAINTS REMOVED ARE LABELED WITH †. THE BEST RESULTS ARE MARKED IN **BOLD**, AND THE SECOND-BEST RESULTS ARE INDICATED IN BROWN.

Design	Xplace				DREAMPlace				XplaceDM				DREAMPlaceDM				Xplace-NN (DM)			
	HPWL	OVFL-5	GP/s	PL/s	HPWL	OVFL-5	GP/s	PL/s	HPWL	OVFL-5	GP/s	PL/s	HPWL	OVFL-5	GP/s	PL/s	HPWL	OVFL-5	GP/s	PL/s
des_perf_1	1106.5	**64.35**	**1.16**	**6.27**	1107.3	65.33	4.07	11.76	**1106.4**	64.82	1.29	7.4	1107.0	65.59	6.32	14.53	1113.7	64.99	2.33	10.71
des_perf_a†	1998.8	52.5	**1.25**	**5.89**	2019.6	53.12	3.83	12.0	**1997.3**	**52.39**	1.38	6.63	2021.5	53.16	5.79	14.34	2023.4	54.52	2.6	7.62
des_perf_b†	1611.8	53.88	**1.32**	**6.02**	**1609.5**	54.53	3.85	11.92	1612.3	53.76	1.42	6.22	1609.7	54.4	5.29	13.87	1614.6	**53.27**	2.47	7.53
edit_dist_a†	4198.3	79.92	**1.45**	**7.13**	4215.7	80.58	4.19	13.97	4198.9	**79.65**	1.55	7.58	4215.7	80.31	5.91	15.84	**4197.5**	80.04	3.11	9.19
fft_1	411.5	56.34	**1.18**	**3.23**	416.1	56.78	4.82	9.45	**411.0**	**55.66**	1.25	3.6	412.1	56.23	5.42	10.08	412.0	56.63	2.48	4.99
fft_2	374.1	47.68	**1.18**	3.48	**372.7**	47.54	3.72	8.16	373.7	**47.52**	1.31	**3.47**	374.3	**47.52**	6.64	11.0	374.3	47.85	2.41	4.79
fft_a	**625.8**	34.8	**1.3**	**3.34**	629.3	35.27	3.67	7.78	626.4	**34.7**	1.39	3.52	628.9	34.98	5.68	10.01	626.9	34.89	2.24	4.76
fft_b	845.6	**51.8**	1.72	**3.77**	845.2	51.98	3.71	7.92	847.1	52.15	**1.43**	3.77	**845.0**	51.89	6.53	11.06	846.3	51.87	2.26	4.53
matrix_mult_1	2116.3	82.19	1.61	7.45	2129.7	82.18	3.65	14.0	**2115.6**	**81.85**	**1.33**	**7.17**	2129.1	81.99	5.07	15.28	2123.0	82.02	2.58	8.48
matrix_mult_2	2152.7	77.53	**1.29**	**6.98**	2164.3	77.47	4.32	14.74	2152.7	76.78	1.41	7.4	2163.2	77.15	7.13	17.54	**2151.2**	**75.91**	2.57	8.58
matrix_mult_a	3032.6	48.21	1.73	9.21	3036.6	**47.91**	4.42	16.12	3032.3	48.32	**1.52**	**8.8**	3037.0	47.96	6.67	18.14	**3030.5**	48.22	2.64	9.68
matrix_mult_b†	2762.6	45.05	1.65	7.72	2787.2	45.02	4.19	14.76	2762.7	45.01	**1.4**	**7.62**	2787.4	45.17	6.43	17.67	**2761.3**	**44.21**	2.47	8.9
matrix_mult_c†	2674.6	42.31	1.74	**7.47**	**2673.1**	42.31	4.26	16.21	2675.6	**42.19**	**1.46**	7.59	2674.0	42.24	5.9	18.37	2677.9	42.31	2.56	8.52
pci_bridge32_a†	360.9	30.65	1.6	3.8	361.8	30.41	3.87	8.12	361.3	30.51	**1.31**	**3.37**	362.0	**30.17**	5.36	9.86	**355.2**	30.45	2.4	4.75
pci_bridge32_b†	**714.0**	23.16	1.49	3.67	741.5	22.79	6.43	11.31	715.1	23.04	**1.23**	**3.64**	739.7	**22.49**	10.08	14.7	714.4	22.61	2.14	4.58
superblue11_a†	33521.3	54.46	**2.94**	**39.74**	**33397.5**	54.5	5.64	73.13	33521.8	54.58	3.95	42.24	33413.7	**54.33**	7.61	74.65	33526.4	54.64	4.78	42.75
superblue12	25784.5	92.44	**4.72**	**54.77**	25799.8	92.62	8.85	91.02	25792.8	**92.27**	5.35	58.17	25791.8	93.24	14.32	98.62	**25694.8**	92.68	8.1	58.76
superblue14	**22776.7**	63.72	**2.01**	**28.84**	23028.4	**63.14**	4.61	52.59	22777.5	63.77	2.12	29.1	23019.6	63.41	7.31	55.14	22784.5	63.26	3.19	30.54
superblue16_a†	25491.0	65.98	**2.19**	**29.42**	25605.4	65.98	4.25	54.93	25500.4	66.03	2.51	30.82	25599.5	65.97	6.64	58.91	**25455.6**	**65.89**	3.63	32.14
superblue19	15542.4	62.32	**1.97**	**22.45**	15630.6	61.81	4.64	42.1	15545.5	62.39	2.13	22.77	15630.7	61.89	6.83	45.73	**15446.4**	**60.88**	3.84	24.83
Mean	7405.1	56.46	**1.78**	**13.03**	7428.6	56.56	4.55	24.6	7406.3	56.37	1.84	13.54	7428.1	56.5	6.85	27.27	**7396.5**	**56.36**	3.04	14.83
Ratio	1.0	1.0	**1.0**	**1.0**	1.0032	1.0018	2.5631	1.8876	1.0002	0.9983	1.0349	1.0392	1.0031	1.0007	3.8572	2.0922	**0.9988**	**0.9981**	1.7127	1.138

and shows state-of-the-art solution quality and performance compared to previous CPU-based global placers. Therefore we compare our Xplace with DREAMPlace[3] on the ISPD 2005 contest benchmarks [36] and ISPD 2015 contest benchmarks [37]. Statistics of benchmarks are given in Table I.

Note that the ISPD 2015 contest benchmarks [37] are for detailed-routability-driven placement. However, the contest-provided evaluation tool is currently inaccessible. Luckily, we find that some commercial tools (e.g. Innovus [38]) can run detailed routing after fixing some errors. In this work, all experiments on ISPD 2015 contest benchmarks are conducted on this fixed version. The fixed version of ISPD 2015 contest benchmarks is released on Xplace's GitHub repository[4].

In Section VIII-A, we experimentally verify Xplace on the ISPD 2005 as well as the ISPD 2015 contest benchmarks and compare Xplace's performance and quality with DREAMPlace. In Section VIII-B, we conduct ablation studies to show the efficiency of our proposed operator-level optimization techniques mentioned in Section IV. In Section VIII-C, we show the effectiveness of our deterministic version. In Section VIII-D, a novel and well-designed neural network is plugged into the framework of Xplace. In Section VIII-E, we evaluate the detailed-routability of Xplace-Route on the ISPD 2015 contest benchmarks and demonstrate the effectiveness of our proposed routability optimization techniques.

A. Validation on Contest Benchmarks

1) ISPD 2005 contest benchmarks: Quantitative results on the ISPD 2005 contest benchmarks are presented in Table II. We compare the HPWL, the global placement time and the total placement time (including I/O, legalization and detailed placement time) with DREAMPlace. "GP/s" and "PL/s" refer to the GP time and total placement time respectively. The best results among all the placers are highlighted in **bold**, and

[3]https://github.com/limbo018/DREAMPlace/tree/b31e8afa60. We observe this version of DREAMPlace's source codes hasn't yet optimized their CUDA kernel launch bound parameters for the NVIDIA Ampere GPU architecture which RTX 3090 is powered by. For fair comparison, we optimize the launch bound parameters in DREAMPlace's source code and report the experimental results based on the modified version.

[4]https://github.com/cuhk-eda/Xplace/tree/main/data

TABLE IV

ABLATION STUDIES OF THE OPERATOR-LEVEL OPTIMIZATION TECHNIQUES ON THE ISPD 2005 BENCHMARKS [36]. OR, OC, OE, OS REFER TO OPERATOR REDUCTION, OPERATOR COMBINATION, OPERATOR EXTRACTION, AND OPERATOR SKIPPING RESPECTIVELY. NOTE THAT XPLACE ENABLES ALL THE OPERATOR-LEVEL OPTIMIZATION TECHNIQUES IN SECTION IV.

Methods	OR	OC	OE	OS	adaptec1	adaptec2	adaptec3	adaptec4	bigblue1	bigblue2	bigblue3	bigblue4	Avg
Ratio	-	-	-	-	234%	194%	136%	124%	198%	140%	123%	121%	159%
	✓	-	-	-	110%	109%	113%	115%	105%	115%	119%	118%	113%
	✓	✓	-	-	107%	107%	107%	108%	104%	108%	113%	112%	108%
	✓	✓	✓	-	104%	102%	104%	104%	102%	104%	106%	105%	104%
Xplace	Ratio				100%	100%	100%	100%	100%	100%	100%	100%	100%
	GP / Iter Time (ms)				1.478	1.671	2.325	2.688	1.572	2.441	4.974	10.018	-
DREAMPlace	Ratio				462%	345%	288%	254%	376%	288%	199%	158%	296%
	GP / Iter Time (ms)				6.832	5.769	6.699	6.840	5.915	7.023	9.904	15.831	-

the second-best results are indicated in brown. Experimental results show that Xplace (without NN) achieves around 1.7x GP time speedup over DREAMPlace and produces better solution quality.

Different from the DAC version [26] that used NTUPlace3 [10], this extension uses ABCDPlace [34] as the default legalizer and detailed placer for both DREAMPlace's and Xplace's GP solutions. Besides, we re-run all the experiments in this extension. Thus the quantitative results in Table II are slightly different from that in the previous DAC version [26].

2) ISPD 2015 contest benchmarks: Quantitative results on the ISPD 2015 contest benchmarks are presented in Table III. Similar to [20], we use NTUplace4dr [12] and the embedded NCTUgr [31] to report the solution quality (HPWL) and the global routing (GR) routability (top5 overflow). Note that top5 overflow measures routability by taking the average overflow of the top 5% most congested GR gcells. Experimental results show that Xplace obtains around 2.6x GP time speedup than DREAMPlace and produces better HPWL with comparable top5 overflow. Considering both ISPD 2005 and ISPD 2015 benchmarks together, Xplace can achieve around 2x GP runtime speedup over DREAMPlace.

B. Ablation Studies on Operator-Level Optimization

We perform ablation studies of our proposed operator-level optimization techniques to demonstrate their effectiveness. We measure the runtime performance by the per global iteration time. Quantitative results are shown in Table IV. It is worth noting that operator combination, operator extraction and operator skipping techniques mainly boost the runtime performance of the larger cases while the operator reduction technique accelerates the smaller cases. These results also show that Xplace can achieve around 3x per GP iteration time speedup compared with DREAMPlace on average.

C. The Effectiveness of Determinism Implementation

We verify the solution quality and runtime performance of Xplace's deterministic version (shortened as XplaceDM) on ISPD 2005 and 2015 contest benchmarks. The results are shown in Table II and Table III. Considering both ISPD 2005 and ISPD 2015 benchmarks together, XplaceDM only needs around 10% runtime overhead to support determinism with comparable solution quality. Compared to the deterministic version of DREAMPlace (denoted as DREAMPlaceDM), XplaceDM is accelerated around 3x in terms of GP time. With the well-designed parallelization of our operator-level optimization, XplaceDM also achieves around 2x GP time speedup over the non-deterministic DREAMPlace while keeping the determinism. The results successfully demonstrate the effectiveness of our determinism implementation.

D. Neural-Enhanced Performance

The proposed model is a lightweight neural network with 471k parameters, which is only 60% of the well-known image-to-image model U-Net [39]. We perform the training based on ISPD 2005 contest benchmarks with their respective macro layouts. The density map and electric fields are used as training data and labels at every iteration with 256×256 resolution. The training scheme is given as follows,

1) Randomly select one of the macro layouts.
2) Randomly generate standard cells at a starting position and push them all over the map with only the density objective $D(p)$ for 40 iterations
3) Repeat (1) and (2) 200 times to generate 8000 samples
4) Train the model on these samples 250 iterations with 128 batch size
5) Consider (1)-(4) as one epoch and repeat 50 epochs.

Recall that the trained model has a good global view to spread cells. We explore embedding this model into our deterministic version XplaceDM. We conduct experiments using the ISPD 2005 and 2015 benchmarks to evaluate Xplace-NN. The results, as shown in Table II and Table III, reveal that Xplace-NN achieves modest improvements of 0.13% and 0.12% on the ISPD 2005 and 2015 benchmarks, respectively.

Although the improvements in HPWL achieved by Xplace-NN may appear marginal, these experimental results still demonstrate the capacity of Xplace to smoothly integrate a neural network. In the future, we believe that the NN integration paradigm outlined in Section VI may prove particularly advantageous for enhancing analytical placers to tackle more complex objectives, especially those that are not straightfor-

TABLE V
DR METRICS AND RUNTIME RESULTS ON THE ISPD 2015 CONTEST BENCHMARKS [37]. "DM" DENOTES DETERMINISM AND "NONDM" DENOTES NON-DETERMINISM. THE BENCHMARKS WITH FENCE REGION CONSTRAINTS REMOVED ARE LABELED WITH †.

Design	**Xplace-Route (DM)**					DREAMPlace (NonDM)					NTUplace4dr (DM)				
	DR WL/um	#DR Vias	#DRVs	PL/s	∗DR/s	DR WL/um	#DR Vias	#DRVs	PL/s	∗DR/s	DR WL/um	#DR Vias	#DRVs	PL/s	∗DR/s
des_perf_1	1439840	564970	10306	10	1801	1449286	570616	19945	12	2391	1509013	630344	2931	357	2721
des_perf_a†	2488402	575632	43854	23	645	2366671	565391	32381	12	652	2397042	569311	2602	302	4480
des_perf_b†	1802951	543049	1323	10	415	1810931	555033	14920	12	2018	1769864	551066	1887	332	468
edit_dist_a†	5750598	1015469	422480	26	2521	5687881	1012545	426136	14	2640	6576184	1125780	1188119	457	5736
fft_1	513869	185766	3478	13	769	522764	188497	8011	9	1375	572738	199139	1397	91	1376
fft_2	621300	191162	1186	12	953	599272	187689	8886	8	423	729743	195227	993	99	753
fft_a	1137087	192259	781	13	1127	1079780	192945	4533	8	1569	1176459	199888	1888	96	2668
fft_b	1282342	213953	16661	14	941	1252724	211800	21013	8	895	1267726	208975	39082	111	835
matrix_mult_1	2648037	828540	12373	26	6933	2712367	812150	79049	14	1344	2699517	892013	4186	339	3517
matrix_mult_2	2678569	857402	11421	26	8809	2723576	842283	69799	15	1482	2733713	906449	5062	360	8217
matrix_mult_a	3941724	848261	7054	12	6536	3881128	864485	26127	16	5073	4190876	869526	4089	381	12186
matrix_mult_b†	3649667	782850	45894	10	1301	3670952	789352	73368	15	1280	3921403	804760	61470	318	1188
matrix_mult_c†	3685351	791702	7518	10	3952	3712692	816093	26761	16	6080	4331793	855656	3784	340	5535
pci_bridge32_a†	651346	145663	4001	5	3163	650836	149042	5969	8	2649	606431	143725	1729	125	2022
pci_bridge32_b†	1007617	147953	347	5	162	1012909	150805	2158	11	320	811280	137266	426	88	119
superblue11_a†	40316503	5659845	1202	78	6906	39973449	5645598	1182	73	6953	64214794	7218914	533505	14321	16325
superblue12	42780736	10508482	29569	311	22614	42477241	11254375	3283142	91	27160	48273446	11768924	31293	7493	24229
superblue14	27959928	4341696	366	70	11901	28386548	4435721	418	53	14639	31959227	5340941	1343	3166	11635
superblue16_a†	31460619	4644486	4489	82	16539	31455744	4732382	3871	55	15284	36908305	5632423	14586	3654	30615
superblue19	20451060	3582347	8255	58	9310	20556809	3612264	7266	42	7285	22402103	4035778	8041	3245	7839
Mean	9813377	1831074	31628	41	5365	9799178	1879453	205747	25	5076	11952583	2114305	95421	1784	7123
Ratio	1.00	1.00	1.00	1.00	1.00	1.00	1.03	6.51	0.60	0.95	1.22	1.15	3.02	43.82	1.33

Design	Xplace (DM)					DREAMPlace + CUGR (NonDM)					Enhanced DREAMPlace + CUGR (NonDM)				
	DR WL/um	#DR Vias	#DRVs	PL/s	∗DR/s	DR WL/um	#DR Vias	#DRVs	PL/s	∗DR/s	DR WL/um	#DR Vias	#DRVs	PL/s	∗DR/s
des_perf_1	1447376	569167	20064	7	2726	1447442	568831	19829	204	2400	1440466	561512	19976	205	2499
des_perf_a†	2338413	558472	28028	7	589	2866082	605591	240085	115	1181	2366966	565054	33102	218	629
des_perf_b†	1816033	557017	14768	6	2310	1828936	566050	14253	173	1937	1820022	546426	13966	251	1901
edit_dist_a†	5671918	1008135	405727	8	2820	6045810	1059270	608067	164	2985	5838307	1080238	493280	202	2876
fft_1	517999	187120	7802	4	1291	516922	188519	8157	44	1317	524240	188222	8485	97	1316
fft_2	598808	188353	9200	3	394	608394	196025	5552	65	1739	629227	194278	6105	89	1552
fft_a	1093206	193126	4677	4	1476	1345219	219195	7641	49	1277	1076563	192419	4855	86	1585
fft_b	1249450	203681	32851	4	928	1405437	224950	44052	41	1224	1255215	211532	21385	90	1090
matrix_mult_1	2703662	807649	76976	7	1306	2711580	821025	79519	84	1318	2722410	812788	79271	164	1325
matrix_mult_2	2718941	840570	68624	7	1694	2710509	845701	56748	125	1681	2709817	833468	54706	160	1655
matrix_mult_a	3877049	861600	25646	9	5540	4708376	914328	41858	113	1400	3892047	865240	25770	154	5268
matrix_mult_b†	3657963	791901	66643	8	1327	4465439	864227	77742	112	1332	3674444	785578	47308	147	1178
matrix_mult_c†	3725954	814420	26195	8	6008	4831883	919329	26265	115	8360	3706182	816044	26340	144	5857
pci_bridge32_a†	642818	148475	5631	3	3130	653934	149437	5941	16	2786	646469	148451	5613	106	2877
pci_bridge32_b†	981717	149362	2076	4	276	1029276	152185	2123	23	350	1028863	151654	2148	127	338
superblue11_a†	40316503	5659845	1202	42	7399	45092596	5907673	1314	1043	7428	40335827	5637829	967	1224	6464
superblue12	42734765	10840792	3122560	58	25770	45921282	11068935	21863	2020	18623	46529015	11083954	15744	2477	21085
superblue14	27955769	4339771	347	29	13082	28925588	4234243	415	943	5143	28548859	4184970	365	1121	5816
superblue16_a†	31460619	4644486	4489	31	16771	31953926	4634390	2495	677	12834	31082724	4568077	2410	815	12601
superblue19	20468106	3584954	8289	23	9165	22266531	3709667	6116	869	5766	22642721	3733021	19036	1326	55805
Mean	9798853	1847445	196590	14	5200	10566758	1892479	63502	350	4054	10123519	1858038	44042	460	6686
Ratio	1.00	1.01	6.22	0.33	0.97	1.08	1.03	2.01	8.59	0.76	1.03	1.01	1.39	11.30	1.25

We use Innovus to run detailed routing under the same setting on the same machine and report DR WL, #DR Vias, #DRVs and DR time.
∗ The DR time (DR/s) may not precisely reflect the placement routability because Innovus will early terminate its detailed router if a design has low routability.

wardly formulated, such as power, thermal management, and so on.

E. Validation on Detailed Routability

1) Experimental Settings: As mentioned in Section VIII, Innovus can be used as an evaluation tool for ISPD 2015 detailed-routability-driven placement contest benchmarks [37]. In this subsection, we will evaluate the detailed routability of Xplace-Route, Xplace, DREAMPlace, routability-driven DREAMPlace. Note that the routability-driven DREAMPlace [1] released on GitHub uses NCTUgr [31] to calculate the congestion map and compute the cell inflation ratio. However, the version of NCTUgr used by DREAMPlace does not support ISPD 2015 LEF/DEF format. Therefore we use the state-of-the-art open-sourced global router CUGR [30] instead[5]. Besides, we also compare Xplace-Route with the state-of-the-art CPU-based detailed-routability-driven placer NTUplace4dr [12].

To verify the placement detailed-routability, we first execute different placers to generate their placement DEF files. Then we use Innovus 20.14 to load the placed solutions and perform detailed routing on a Linux machine with 4x Intel Xeon E7-4830 v2 (2.20GHz). For a fair comparison, we use the same settings and parameters in Innovus to run detailed routing on these placement solutions with 10 threads enabled[6]. We

[5]https://github.com/cuhk-eda/cu-gr
[6]The DR evaluation script is also provided in Xplace's GitHub Repository https://github.com/cuhk-eda/Xplace/tree/main/tool/innovus_ispd2015_fix.

TABLE VI
EFFECTIVENESS OF OUR PIN-ACCESSIBILITY POST-REFINEMENT (PA-RF). IMPROVED BY PA-RF IS MARKED IN LIGHT BLUE.

Placer	PA-RF	DR WL/um		#DR Vias		#DRVs	
		Mean	Ratio	Mean	Ratio	Mean	Ratio
Xplace-Route (Ours)	✓	9813377	1.00	1831074	1.00	**31628**	**1.00**
DREAMPlace	-	9799178	1.00	1879453	1.03	205747	6.51
	✓	9799131	1.00	1877990	1.03	201066	6.36
NTUplace4dr	-	11952583	1.22	2114305	1.15	95421	3.02
	✓	11952282	1.22	2114190	1.15	95417	3.02
Xplace	-	9798853	1.00	1847445	1.01	196590	6.22
	✓	9798200	1.00	1846346	1.01	190759	6.03
DREAMPlace + CUGR	-	10566758	1.08	1892479	1.03	63502	2.01
	✓	10565582	1.08	1892014	1.03	58684	1.86
Enhanced DREAMPlace + CUGR	-	10123519	1.03	1858038	1.01	44042	1.39
	✓	10124168	1.03	1856316	1.01	40064	1.27

TABLE VII
ABLATION STUDIES OF PROPOSED ROUTABILITY OPTIMIZATION TECHNIQUES ON THE ISPD 2015 CONTEST BENCHMARKS [37]. CL, D_{m2rail}, D_{iopin}, AND PA-RF REFER TO CELL INFLATION, M2 RAIL DENSITY INSERTION, I/O PIN DENSITY INCREMENT, AND PIN-ACCESSIBILITY POST-REFINEMENT RESPECTIVELY. NOTE THAT XPLACE-ROUTE ENABLES ALL THE DETAILED-ROUTABILITY OPTIMIZATION TECHNIQUES IN SECTION VII.

Methods				DR WL/um		#DR Vias		#DRVs	
CL	D_{m2rail}	D_{iopin}	PA-RF	Mean	Ratio	Mean	Ratio	Mean	Ratio
-	-	-	-	9798853	0.999	1847445	1.009	196590	6.216
✓	-	-	-	9796424	0.998	1833652	1.001	37787	1.195
✓	✓	-	-	9808884	1.000	1831921	1.000	34355	1.086
✓	✓	✓	-	9813084	1.000	1832366	1.001	34002	1.075
Xplace-Route (Ours)				9813377	1.000	1831074	1.000	31628	1.000

measure the routability of a placement solution by the following metrics: the detailed routing wirelength (DR WL/um), the number of DR vias (#DR Vias), the number of detailed routing violations (#DRVs), and the placement time (PL/s).

2) Quantitative Results: The results are shown in Table V. Xplace-Route runs in deterministic mode and successfully outperforms the other baselines in terms of detailed routing metrics. Compared to the original Xplace and DREAMPlace, Xplace-Route remarkably boosts the detailed-routability, especially #DRVs, while only taking no more than 27 seconds (on average) extra runtime overhead. Compared to the routability-driven DREAMPlace (i.e. DREAMPlace + CUGR) and NTUplace4dr, Xplace-Route shows better routability with a much shorter placement elapsed time. The better solution quality demonstrates the necessity of detailed-routing-aware techniques, and the shorter elapsed time indicates the effectiveness of our GPU-accelerated scheme.

For reference, we also report the DR time (DR/s) in Table V. However, the DR runtime may not precisely reflect the routability. For example, compared to the routability-driven DREAMPlace's solution, the commercial detailed router on Xplace-Route's solution uses shorter WL, fewer vias, and fewer DRVs but uses more runtime. The reason is that the commercial detailed router will be early terminated if a design has low routability. Therefore, we choose to measure the detailed routability by DR WL, #DR Vias, and #DRVs instead of DR time.

To measure our flow efficiency, we embed our re-initializing strategy and solution selection criteria discussed in Section VII-C and Section VII-D into the routability-driven DREAMPlace (denoted as Enhanced DREAMPlace + CUGR in Table V). The enhanced version achieves a visible routability improvement and only needs 30% extra runtime overhead compared to the original routability-driven DREAMPlace. The results demonstrate the effectiveness of our re-initialization technique and solution selection criteria.

To demonstrate the effectiveness of our pin-accessibility post-refinement technique in Section VII-E, we apply it to the other baselines and conduct ablation studies. As shown in Table VI, PA-refine reduces the average #DRVs and improves the detailed routability for all the other placers. It is worth noting that despite the improvements brought about by PA-refine in these placers, Xplace-Route continues to exhibit superior routability performance compared to these enhanced results. This further indicates the efficiency of our routability optimization flow.

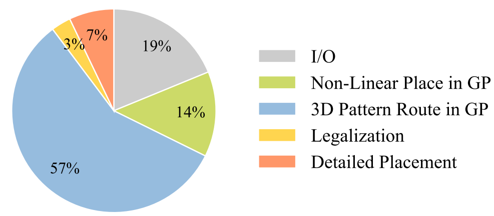


Fig. 7. Runtime breakdown of Xplace-Route on ISPD 2015 benchmarks.

We also conduct ablation studies on our proposed detailed-routability optimization techniques. Table VII lists the average DR WL, #DR Vias and #DRVs of different experimental settings. The results demonstrate the efficiency of the proposed detailed-routability-driven optimization techniques in Xplace-Route. Concretely, cell inflation, M2 rail density insertion, I/O pin density increment, and pin-accessibility post-refinement achieve around 502%, 11%, 1%, and 8% #DRVs reduction respectively.

3) Runtime breakdown: Fig. 7 shows the runtime breakdown of Xplace-Route on ISPD 2015 contest benchmarks. We measure the average runtime of different parts among all the cases in ISPD 2015 benchmarks, including I/O time, non-linear placement time in GP, 3D pattern routing time in GP, legalization time, and detailed placement time. We calculate their proportion to the runtime of routability-driven placement. The results show that our non-linear GPU-accelerated placement optimization only takes 14% of the runtime. From the percentage perspective, the pattern router in GP is the most time-consuming part and takes 57% of the runtime. However, from the absolute value perspective, the routing part only takes 23 seconds (the runtime of Xplace-Route is 41 seconds), which is still very effective.

IX. CONCLUSIONS

In this work, we propose Xplace, a new, fast, extensible and open-source GPU-accelerated placement framework. We also extend Xplace with neural enhancement and routability optimization. Experimental results on the ISPD 2005 and the

ISPD 2015 benchmarks show that Xplace is efficient yet extensible.

In the future, we will include handling additional complex constraints in placement, such as fence region and timing. We plan to combine Xplace-Route with DNNs to further enhance the detailed-routability. We also plan to extend the integration paradigm of Xplace-NN to optimize more intricate objectives like power.

ACKNOWLEDGMENTS

The authors would like to thank Prof. Yibo Lin and the other authors of DREAMPlace [1], [20], [21] and ABCDPlace [34] for releasing the code on GitHub, and Prof. Yao-Wen Chang and the other authors of NTUplace4dr [12] for sharing the binary.

REFERENCES

[1] Y. Lin, Z. Jiang, J. Gu, W. Li, S. Dhar, H. Ren, B. Khailany, and D. Z. Pan, "Dreamplace: Deep learning toolkit-enabled gpu acceleration for modern vlsi placement," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 4, pp. 748–761, 2021.

[2] X. He, T. Huang, L. Xiao, H. Tian, and E. F. Y. Young, "Ripple: A robust and effective routability-driven placer," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 10, pp. 1546–1556, 2013.

[3] N. K. Darav, A. Kennings, A. F. Tabrizi, D. Westwick, and L. Behjat, "Eh?placer: A high-performance modern technology-driven placer," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 21, no. 3, apr 2016.

[4] T. Lin, C. Chu, and G. Wu, "Polar 3.0: An ultrafast global placement engine," in *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2015, pp. 520–527.

[5] M.-C. Kim, D.-J. Lee, and I. L. Markov, "Simpl: An effective placement algorithm," in *2010 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2010, pp. 649–656.

[6] P. Spindler, U. Schlichtmann, and F. M. Johannes, "Kraftwerk2—a fast force-directed quadratic placement approach using an accurate net model," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 8, pp. 1398–1411, 2008.

[7] M.-C. Kim and I. L. Markov, "Complx: A competitive primal-dual lagrange optimization for global placement," in *Proceedings of the 49th Annual Design Automation Conference*. New York, NY, USA: Association for Computing Machinery, 2012, p. 747–752.

[8] N. Viswanathan, M. Pan, and C. Chu, "Fastplace 3.0: A fast multilevel quadratic placement algorithm with placement congestion control," in *Proceedings of the 2007 Asia and South Pacific Design Automation Conference*. USA: IEEE Computer Society, 2007, p. 135–140.

[9] A. Kahng and Q. Wang, "Implementation and extensibility of an analytic placer," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 5, pp. 734–747, 2005.

[10] T.-C. Chen, Z.-W. Jiang, T.-C. Hsu, H.-C. Chen, and Y.-W. Chang, "Ntuplace3: An analytical placer for large-scale mixed-size designs with preplaced blocks and density constraints," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 7, pp. 1228–1240, 2008.

[11] M.-K. Hsu, Y.-F. Chen, C.-C. Huang, S. Chou, T.-H. Lin, T.-C. Chen, and Y.-W. Chang, "Ntuplace4h: A novel routability-driven placement algorithm for hierarchical mixed-size circuit designs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, no. 12, pp. 1914–1927, 2014.

[12] C.-C. Huang, H.-Y. Lee, B.-Q. Lin, S.-W. Yang, C.-H. Chang, S.-T. Chen, Y.-W. Chang, T.-C. Chen, and I. Bustany, "Ntuplace4dr: A detailed-routing-driven placer for mixed-size circuit designs with technology and region constraints," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 3, pp. 669–681, 2018.

[13] T. Chan, J. Cong, and K. Sze, "Multilevel generalized force-directed method for circuit placement," in *Proceedings of the 2005 International Symposium on Physical Design*. New York, NY, USA: Association for Computing Machinery, 2005, p. 185–192.

[14] W. Zhu, Z. Huang, J. Chen, and Y.-W. Chang, "Analytical solution of poisson's equation and its application to vlsi global placement," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2018, pp. 1–8.

[15] J. Lu, H. Zhuang, P. Chen, H. Chang, C.-C. Chang, Y.-C. Wong, L. Sha, D. Huang, Y. Luo, C.-C. Teng, and C.-K. Cheng, "eplace-ms: Electrostatics-based placement for mixed-size circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 5, pp. 685–698, 2015.

[16] J. Lu, P. Chen, C.-C. Chang, L. Sha, D. J.-H. Huang, C.-C. Teng, and C.-K. Cheng, "Eplace: Electrostatics-based placement using fast fourier transform and nesterov's method," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 20, no. 2, mar 2015.

[17] C.-K. Cheng, A. B. Kahng, I. Kang, and L. Wang, "Replace: Advancing solution quality and routability validation in global placement," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 9, pp. 1717–1730, 2019.

[18] J. Cong and Y. Zou, "Parallel multi-level analytical global placement on graphics processing units," in *Proceedings of the 2009 International Conference on Computer-Aided Design*. New York, NY, USA: Association for Computing Machinery, 2009, p. 681–688.

[19] C.-X. Lin and M. D. F. Wong, "Accelerate analytical placement with gpu: A generic approach," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2018, pp. 1345–1350.

[20] J. Gu, Z. Jiang, Y. Lin, and D. Z. Pan, "Dreamplace 3.0: Multi-electrostatics based robust vlsi placement with region constraints," in *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2020, pp. 1–9.

[21] P. Liao, D. Guo, Z. Guo, S. Liu, Y. Lin, and B. Yu, "Dreamplace 4.0: Timing-driven placement with momentum-based net weighting and lagrangian-based refinement," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 10, pp. 3374–3387, 2023.

[22] M.-C. Kim, J. Hu, D.-J. Lee, and I. L. Markov, "A simplr method for routability-driven placement," in *2011 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2011, pp. 67–73.

[23] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. Red Hook, NY, USA: Curran Associates Inc., 2019.

[24] M.-K. Hsu, Y.-W. Chang, and V. Balabanov, "Tsv-aware analytical placement for 3d ic designs," in *Proceedings of the 48th Design Automation Conference*. New York, NY, USA: Association for Computing Machinery, 2011, p. 664–669.

[25] S. N. Adya, I. L. Markov, and P. G. Villarrubia, "On whitespace and stability in mixed-size placement and physical synthesis," in *Proceedings of the 2003 IEEE/ACM International Conference on Computer-Aided Design*. USA: IEEE Computer Society, 2003, p. 311.

[26] L. Liu, B. Fu, M. D. F. Wong, and E. F. Y. Young, "Xplace: An extremely fast and extensible global placement framework," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*. New York, NY, USA: Association for Computing Machinery, 2022, p. 1309–1314.

[27] T. Lin, C. Chu, J. R. Shinnerl, I. Bustany, and I. Nedelchev, "Polar: Placement based on novel rough legalization and refinement," in *2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2013, pp. 357–362.

[28] Z. Li, N. B. Kovachki, K. Azizzadenesheli, B. liu, K. Bhattacharya, A. Stuart, and A. Anandkumar, "Fourier neural operator for parametric partial differential equations," in *International Conference on Learning Representations*, 2021.

[29] S. Lin and M. D. Wong, "Superfast full-scale gpu-accelerated global routing," in *2022 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2022, pp. 1–8.

[30] J. Liu, C.-W. Pui, F. Wang, and E. F. Y. Young, "Cugr: Detailed-routability-driven 3d global routing with probabilistic resource model," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6.

[31] W.-H. Liu, W.-C. Kao, Y.-L. Li, and K.-Y. Chao, "Nctu-gr 2.0: Multithreaded collision-aware global routing with bounded-length maze routing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 5, pp. 709–722, 2013.

[32] J. A. Roy, N. Viswanathan, G.-J. Nam, C. J. Alpert, and I. L. Markov, "Crisp: Congestion reduction by iterated spreading during placement," in *Proceedings of the 2009 International Conference on Computer-Aided*

Design. New York, NY, USA: Association for Computing Machinery, 2009, p. 357–362.

[33] Y. Wei, C. Sze, N. Viswanathan, Z. Li, C. J. Alpert, L. Reddy, A. D. Huber, G. E. Tellez, D. Keller, and S. S. Sapatnekar, "Glare: Global and local wiring aware routability evaluation," in *Proceedings of the 49th Annual Design Automation Conference*. New York, NY, USA: Association for Computing Machinery, 2012, p. 768–773.

[34] Y. Lin, W. Li, J. Gu, H. Ren, B. Khailany, and D. Z. Pan, "Abcdplace: Accelerated batch-based concurrent detailed placement on multithreaded cpus and gpus," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 12, pp. 5083–5096, 2020.

[35] H. Li, W.-K. Chow, G. Chen, B. Yu, and E. F. Young, "Pin-accessible legalization for mixed-cell-height circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 1, pp. 143–154, 2022.

[36] G.-J. Nam, C. J. Alpert, P. Villarrubia, B. Winter, and M. Yildiz, "The ispd2005 placement contest and benchmark suite," in *Proceedings of the 2005 International Symposium on Physical Design*. New York, NY, USA: Association for Computing Machinery, 2005, p. 216–220.

[37] I. S. Bustany, D. Chinnery, J. R. Shinnerl, and V. Yutsis, "Ispd 2015 benchmarks with fence regions and routing blockages for detailed-routing-driven placement," in *Proceedings of the 2015 Symposium on International Symposium on Physical Design*. New York, NY, USA: Association for Computing Machinery, 2015, p. 157–164.

[38] "Cadence innovus implementation system," https://www.cadence.com.

[39] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention - MICCAI 2015 - 18th International Conference Munich, Germany, October 5 - 9, 2015, Proceedings, Part III*, vol. 9351. Springer, 2015, pp. 234–241.

Lixin Liu received his B.Eng. degree from Electronics Science and Technology, South China University of Technology in 2019, and Ph.D. degree from the Department of Computer Science and Engineering, The Chinese University of Hong Kong in 2023. His research interests include electronic design automation and distributed deep learning systems.

Bangqi Fu received his B.S.E. degree in electronic engineering from Shanghai Jiao Tong University in 2021. He is currently a Ph.D. student at the Department of Computer Science and Engineering, The Chinese University of Hong Kong. His research interests include physical design algorithms and machine learning for Electronic Design Automation, specifically global placement.

Shiju Lin received the B.Eng. degree in computer science and technology from the South China University of Technology, Guangzhou, Guangdong, China, in 2020. 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 GPU acceleration and combinatorial optimization for Electronic Design Automation.

Jinwei Liu received his B.Sc degree in Computer Science and Technology from Sichuan University, Sichuan, China, in 2018, and Ph.D. degree from the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong, in 2022. His research interests include electronic design automation, combinatorial optimization and machine learning.

Evangeline F.Y. Young (Fellow, IEEE) received the B.Sc. degree in computer science from The Chinese University of Hong Kong (CUHK), Hong Kong, and the Ph.D. degree from The University of Texas at Austin, Austin, TX, USA, in 1999.

She is currently a Professor with the Department of Computer Science and Engineering, CUHK. Her research interests include EDA, optimization, algorithms and AI. Her research focuses on floorplanning, placement, routing, DFM and EDA on physical design in general.

Dr. Young's research group has won championships and prizes in numerous renown EDA contests, including the CAD Contests at ICCAD, DAC and ISPD. She has served on the organization committees of ICCAD and ISPD, and on the program committees of conferences, including DAC, ICCAD, ISPD, DATE and ASP-DAC. She also served on the editorial boards of IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, ACM Transactions on Design Automation of Electronic Systems, and Integration, the VLSI Journal. She is a Fellow of IEEE.

Martin D.F. Wong (Fellow, IEEE) received the B.Sc. degree in mathematics from the University of Toronto, Toronto, ON, Canada, in 1979, and the M.S. degree in mathematics and the Ph.D. degree in computer science (CS) from the University of Illinois at Urbana-Champaign (UIUC), Champaign, IL, USA, in 1981 and 1987, respectively. He was a Faculty Member with The University of Texas at Austin (UT-Austin), Austin, TX, USA, from 1987 to 2002 and UIUC from 2002 to 2018. He was a Bruton Centennial Professor of CS with UT-Austin 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 July 2023, he was with The Chinese University of Hong Kong, Hong Kong, as the Dean of Engineering and the Choh-Ming Li Professor of Computer Science and Engineering. Since August 2023, he joined Hong Kong Baptist University 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 electronic design automation (EDA). His main research interest is in EDA. He is a Fellow of IEEE and ACM.