elfPlace: Electrostatics-based Placement for Large-Scale Heterogeneous FPGAs

Wuxi Li, Yibo Lin, and David Z. Pan
ECE Department, University of Texas at Austin, Austin, Texas, USA
{wuxi.li, yibolin}@utexas.edu; dpan@ece.utexas.edu

***Abstract*—elfPlace is a flat nonlinear placement algorithm for large-scale heterogeneous field-programmable gate arrays (FPGAs). We adopt the analogy between placement and electrostatic systems initially proposed by ePlace and extend it to tackle heterogeneous blocks in FPGA designs. To achieve satisfiable solution quality with fast and robust numerical convergence, an augmented Lagrangian formulation together with a preconditioning technique and a normalized subgradient-based multiplier updating scheme are proposed. Besides pure-wirelength minimization, we also propose a unified instance area adjustment scheme to simultaneously optimize routability, pin density, and downstream clustering compatibility. Our experiments on ISPD 2016 benchmark suite show that elfPlace outperforms four state-of-the-art FPGA placers UTPlaceF, RippleFPGA, GPlace3.0, and UTPlaceF-DL by 13.6%, 11.3%, 8.9%, and 7.1%, respectively, in routed wirelength with competitive runtime.**

I. Introduction

Placement is becoming ever more crucial and challenging due to the drastic evolution of FPGA architecture in the past decades. Modern FPGA has thousands of digital signal processing (DSP) and random-access memory (RAM) blocks and millions of lookup table (LUT) and flip-flop (FF) instances. These heterogeneous resources are often exclusively scattered over discrete locations on the FPGA fabric. This complexity and heterogeneity significantly challenge the effectiveness and efficiency of modern FPGA placers and which play an important role in determining the overall FPGA implementation quality.

There are various core FPGA placement algorithms have been proposed in the literature. Simulated-annealing approaches [1], [2] iteratively perform probabilistic swapping to progressively improve placement solutions. Despite that the global optimum can be reached theoretically, simulated-annealing approaches, in general, suffer from extremely slow convergence. Min-cut approaches [3] distribute instances by recursive netlist partitioning. In spite of performing well on small designs, min-cut approaches often produce unacceptably suboptimal solutions when the design size reaches the scale of millions. Analytical approaches, on the other hand, formulate the entire placement problem as more sophisticated continuous optimization problems. Quadratic approaches [4]–[11] approximate the placement objective using quadratic functions, while nonlinear approaches [12]–[14] use higher-order ones. Compared with quadratic approaches, nonlinear approaches often achieve better solution quality due to their even stronger expressive power.

In contrast to the enormous research endeavor spent on core placement algorithms, there are still very limited works coping with resource heterogeneity issue in FPGA. Most existing analytical placers only treat highly-discrete DSP and RAM blocks specially and eliminate the heterogeneity between LUTs and FFs by either spreading them together with adjusted areas [8], [9], [11] or simply clustering them before placement [13]. These approaches are usually highly sensitive to the heuristics applied, which could hamper the solution quality and placement robustness. A more recent work [15] proposed a multi-commodity flow-based algorithm for quadratic placers to spread heterogeneous instances, and it demonstrated significant improvement over previous spreading heuristics. However, due to the inherent limitation of quadratic placement, their approach still simplifies spreading as a movement-minimization problem, which cannot explicitly optimize wirelength nor preserve the relative order among instances of different resource types.

There are also works on optimizing other placement objectives to ease the downstream clustering, legalization, and routing steps. Many works [8], [9], [11] adopted instance inflation technique to alleviate routing congestions. Li et al. [16] further considered the impact of downstream clustering/legalization and adjusted instance areas accordingly during placement to improve the overall solution quality. However, all these works were originally proposed for quadratic placers, studies on extending them to more powerful nonlinear placement are still lacking.

In this paper, we present elfPlace, a general, flat, nonlinear placement algorithm for large-scale heterogeneous FPGAs. elfPlace adopts the idea of casting placement to electrostatic systems initially proposed by ePlace family [17], [18] for application-specific integrated circuits (ASICs), and it is enhanced to tackle the FPGA heterogeneity issue in a unified and elegant way. Besides the conventional wirelength objective, elfPlace also performs routability, pin density, and clustering-aware optimizations to achieve even higher-quality and smoother design closure. Our major contributions are summarized as follows.

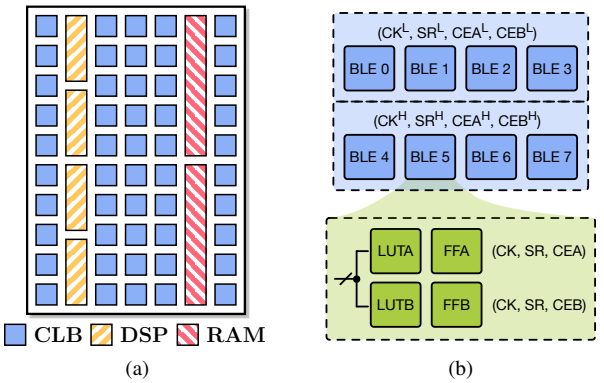
- We enhance the original ePlace algorithm [17], [18] for ASICs to deal with heterogeneous resource types in FPGAs.
- We employ augmented Lagrangian method, instead of the multiplier method used in ePlace, to formulate the nonlinear placement problem.
- We propose a preconditioning technique to improve the numerical convergence given the wide spectrum of instance sizes and net degrees in FPGA designs.
- We propose a normalized subgradient method to update density penalty multipliers, which control the spreading of different resource types in a self-adaptive manner.
- We improve the clustering-aware area adjustment technique proposed in [16] and integrate it, together with routability and pin density optimizations, into elfPlace.
- We demonstrate more than 7% improvement in routed wirelength, on ISPD 2016 benchmark suite [19], over four cutting-edge placers with very competitive runtime.

The rest of this paper is organized as follows. Section II introduces the background knowledge. Section III sketches the overall flow of elfPlace. Section IV describes the core placement algorithms and Section V details the routability, pin density, and clustering-aware optimizations. Section VI shows the experimental results, followed by the conclusion and future work in Section VII.

II. Preliminaries

A. FPGA Architecture

elfPlace is developed based on Xilinx UltraScale [20], which is a representative column-based FPGA architecture that has been also adopted by many other state-of-the-art commercial FPGAs (e.g.,



(a) (b)

Fig. 1: (a) A simplified column-based FPGA architecture. (b) The configurable logic block (CLB) structure.

Xilinx UltraScale+ series). As shown in Figure 1(a), each column in this architecture provides one type of logic resources among configurable logic block (CLB), DSP, and RAM. Columns of different resource types are usually unevenly interleaved over the FPGA fabric. Figure 1(b) details the CLB structure in this architecture, where each CLB consists of 8 basic logic elements (BLEs) and each BLE further contains 2 LUTs and 2 FFs. The 2 LUTs in the same BLE are subject to a maximum input pin count constraint. While FFs in the same CLB are subject to control set constraint. More specifically, as shown in Figure 1(b), a CLB can be divided into two half CLBs, and each of which consists of 4 BLEs that share the same clock (CK), set/reset (SR), and clock enable (CEA/CEB) signals. Therefore, in each half CLB, FFs must share the same CK/SR and FFs with the same polarity (FFA/FFB) must further share the same CE (CEA/CEB).

B. The ePlace Algorithm

`ePlace` [17], [18] is a leading-edge nonlinear global placement algorithm for ASICs. It approximates half-perimeter wirelength (HPWL),

$$W(\boldsymbol{x},\boldsymbol{y}) = \sum_{e\in\mathcal{E}} W_e(\boldsymbol{x},\boldsymbol{y}) = \sum_{e\in\mathcal{E}} \Big(\max_{i,j\in e} |x_i - x_j| + \max_{i,j\in e} |y_i - y_j| \Big), \tag{1}$$

using the weighted-average (WA) model [21], [22],

$$\widetilde{W}_{e_x}(\boldsymbol{x},\boldsymbol{y}) = \frac{\sum_{i\in e} x_i \exp(x_i/\gamma)}{\sum_{i\in e} \exp(x_i/\gamma)} - \frac{\sum_{i\in e} x_i \exp(-x_i/\gamma)}{\sum_{i\in e} \exp(-x_i/\gamma)}. \tag{2}$$

Here $\boldsymbol{x}$ and $\boldsymbol{y}$ denote the instance locations, $\mathcal{E}$ denotes the set of nets in the design, and γ is a parameter to control the modeling smoothness and accuracy. Equation (2) only gives the x-directed WA model of a net and the total wirelength cost is defined as $\widetilde{W}(\boldsymbol{x},\boldsymbol{y}) = \sum_{e\in\mathcal{E}} (\widetilde{W}_{e_x}(\boldsymbol{x},\boldsymbol{y}) + \widetilde{W}_{e_y}(\boldsymbol{x},\boldsymbol{y}))$.

The key innovation of `ePlace` is that it casts the placement density cost to the potential energy of an electrostatic system. With this transformation, each instance i is modeled as a positive charge q_i with the quantity proportional to its area. Given the notations defined in Table I, the electric force $\boldsymbol{F}_i = q_i\boldsymbol{\xi}_i = -q_i\nabla\psi_i$ will guide each charge i towards the direction of minimizing the total potential energy Φ,

$$\Phi(\boldsymbol{x},\boldsymbol{y}) = \iint_R \rho(x,y)\psi(x,y), (x,y) \in R. \tag{3}$$

The unique solution of the electrostatic system is given by Eq. (4).

$$\begin{cases} \nabla\cdot\nabla\psi(x,y) = -\nabla\cdot\boldsymbol{\xi}(x,y) = -\rho(x,y), (x,y)\in R, & \text{(4a)} \\ \hat{\mathbf{n}}\cdot\nabla\psi(x,y) = -\hat{\mathbf{n}}\cdot\boldsymbol{\xi}(x,y) = \mathbf{0}, (x,y)\in \partial R, & \text{(4b)} \\ \iint_R \rho(x,y) = \iint_R \psi(x,y) = 0, (x,y)\in R, & \text{(4c)} \end{cases}$$

TABLE I: Notations used in the electrostatic system

R	A finite two-dimensional region
q_i	The electric charge quantity of charge i
$\rho(x,y)$	The electric charge density at $(x,y)\in R$
$\psi_i, \psi(x,y)$	The electric potential at charge i and $(x,y)\in R$
$\boldsymbol{\xi}_i, \boldsymbol{\xi}(x,y)$	The electric field at charge i and $(x,y)\in R$
$\Phi(\boldsymbol{x},\boldsymbol{y})$	The total electric potential energy of placement $(\boldsymbol{x},\boldsymbol{y})$

where Eq. (4a) is the Poisson's equation to correlate electric potential, electric field, and charge density, Eq. (4b) is Neumann boundary condition (i.e., zero electric field on the boundary of R) to prevent charges from moving out of R, and Eq. (4c) neutralizes the overall electric charge and potential to ensure the solution uniqueness of Eq. (4). `ePlace` honors the placement density constraints by enforcing the electrostatic equilibrium state, where electric density is evenly distributed and $\Phi(\boldsymbol{x},\boldsymbol{y}) = 0$.

`ePlace` computes the numerical solution of Eq. (4) using spectral methods. It divides the placement region into a grid of $m\times m$ bins to construct the charge density map ρ, then the electric potential ψ and electric field $\boldsymbol{\xi} = (\xi_x, \xi_y)$ can be obtained as follows.

$$a_{u,v} = \frac{1}{m^2}\sum_{x=0}^{m-1}\sum_{y=0}^{m-1}\rho(x,y)\cos(\omega_u x)\cos(\omega_v y), \tag{5a}$$

$$\psi(x,y) = \sum_{u=0}^{m-1}\sum_{v=0}^{m-1}\frac{a_{u,v}}{\omega_u^2+\omega_v^2}\cos(\omega_u x)\cos(\omega_v y), \tag{5b}$$

$$\xi_x(x,y) = \sum_{u=0}^{m-1}\sum_{v=0}^{m-1}\frac{a_{u,v}\omega_u}{\omega_u^2+\omega_v^2}\sin(\omega_u x)\cos(\omega_v y), \tag{5c}$$

$$\xi_y(x,y) = \sum_{u=0}^{m-1}\sum_{v=0}^{m-1}\frac{a_{u,v}\omega_v}{\omega_u^2+\omega_v^2}\cos(\omega_u x)\sin(\omega_v y). \tag{5d}$$

Here x and y are bin indexes, u and v denote frequency indexes from 0 to $m-1$, and $\omega_u = \frac{2\pi u}{m}$ and $\omega_v = \frac{2\pi v}{m}$ are the frequencies of $\sin/\cos$ wave functions. Equation (5) can be efficiently computed using discrete cosine transform (DCT) and its inverse (IDCT).

Finally, with both wirelength cost $\widetilde{W}(\boldsymbol{x},\boldsymbol{y})$ and density penalty $\Phi(\boldsymbol{x},\boldsymbol{y})$ well defined, `ePlace` then iteratively solves the following unconstrained nonlinear optimization problem using multiplier method,

$$\min_{\boldsymbol{x},\boldsymbol{y}} f(\boldsymbol{x},\boldsymbol{y}) = \widetilde{W}(\boldsymbol{x},\boldsymbol{y}) + \lambda\Phi(\boldsymbol{x},\boldsymbol{y}), \tag{6}$$

where λ is the density penalty multiplier to progressively enforce the density constraint.

III. elfPlace Overview

One major challenge of FPGA placement is heterogeneity handling. `elfPlace` tackles this problem by maintaining separate electrostatic systems for different resource types, including LUT, FF, DSP, and RAM. The notations used in `elfPlace` are given in Table II.

TABLE II: Notations used in `elfPlace`

$\mathcal{S}$	The resource type set {LUT, FF, DSP, RAM}
$\mathcal{V}$, $\mathcal{V}_s$	The instance set and its subset with resource type s
$\mathcal{V}^{\text{P}}$, $\mathcal{V}_s^{\text{P}}$	The physical instance set and its subset of resource type s
$\mathcal{V}^{\text{F}}$, $\mathcal{V}_s^{\text{F}}$	The filler instance set and its subset of resource type s
A_i	The area of instance i
$\mathcal{B}_s$	The bin grid for resource type $s\in\mathcal{S}$
A_b^{P}	The physical instance area in bin b
C_b	The resource capacity in bin b
$\boldsymbol{\lambda}$	The density multiplier vector $(\lambda_{\text{LUT}}, \lambda_{\text{FF}}, \lambda_{\text{DSP}}, \lambda_{\text{RAM}})^T$
$\boldsymbol{\Phi}$	The potential energy vector $(\Phi_{\text{LUT}}, \Phi_{\text{FF}}, \Phi_{\text{DSP}}, \Phi_{\text{RAM}})^T$

Figure 2 illustrates the overall flow of `elfPlace`. Different from a typical initial placement that minimizes wirelength by quadratic

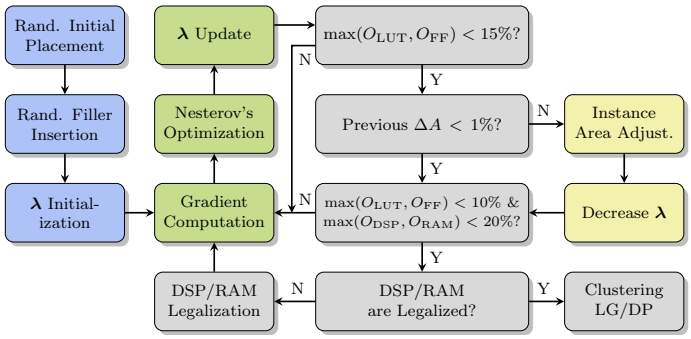


Fig. 2: The overall flow of `elfPlace`.

programming, `elfPlace` starts from a random initial placement, which has been observed to achieve nearly the same quality with considerable runtime reduction [23]. In the random initial placement, all movable instances are first placed at the centroid of fixed pins and an extra Gaussian noise perturbation is injected with standard deviation equal to 0.1% of the width and height of the placement region.

After the initial placement, filler instances are created and inserted independently for each resource type. Fillers are needed to pad whitespaces and produce compact placement solutions. For each resource type $s \in \mathcal{S}$ with the bin grid $\mathcal{B}_s$, its total filler area is computed as $\sum_{b\in\mathcal{B}_s} C_b - \sum_{i\in\mathcal{V}_s^{\text{P}}} A_i$. In our experiments, LUT/FF fillers are set to be squares with $1/8$ CLB area, and DSP and RAM fillers are set to be rectangles with dimensions 1.0×2.5 and 1.0×5.0 (CLB width), respectively, based on the FPGA architecture. For each resource type s, fillers are randomly inserted based on the resource capacity distribution. More specifically, `elfPlace` first randomly distributes fillers of resource type s into bins based on the probabilities $C_b / \sum_{b\in\mathcal{B}_s} C_b$, and their final locations are then uniformly drawn within bins. By this insertion strategy, fillers can start with relatively low potential energy and it improves the convergence and stability of the later placement optimization.

Based on the initial placement and filler insertion, `elfPlace` initializes the density multiplier vector $\boldsymbol{\lambda} = (\lambda_{\text{LUT}}, \lambda_{\text{FF}}, \lambda_{\text{DSP}}, \lambda_{\text{RAM}})^T$ and then enters the core placement optimization phase. In each placement iteration, the gradient of a wirelength-density co-optimization problem is computed and fed to a Nesterov's optimizer [17] to take a descent step. After that, $\boldsymbol{\lambda}$ is updated to balance the spreading efforts on different resource types and universally emphasize slightly more density penalties. When both LUT and FF overflows (O_{LUT} and O_{FF}) are reduced down to 15%, `elfPlace` adjusts instance areas with the consideration of routability, pin density, and downstream clustering compatibility (i.e., LUT input pin constraint and FF control set constraint described in Section II-A). After the instance area adjustment, the nearly-equilibrated electrostatic states are likely to be damaged, therefore, `elfPlace` reduces the density multipliers $\boldsymbol{\lambda}$ in this case to recover the quality again. This area adjustment step is performed each time that LUT/FF converge to $\max(O_{\text{LUT}}, O_{\text{FF}}) < 15\%$ until the total area change is less than 1%. The overflow of each resource type s is given in Eq. (7).

$$O_s = \frac{\sum_{b\in\mathcal{B}_s} \max(A_b^{\text{P}} - C_b, 0)}{\sum_{b\in\mathcal{B}_s} A_b^{\text{P}}}, \forall s \in \mathcal{S}. \tag{7}$$

Once the instance area converges and the overlaps are small enough for all resource types, i.e., $\max(O_{\text{LUT}}, O_{\text{FF}}) < 10\%$ and $\max(O_{\text{DSP}}, O_{\text{RAM}}) < 20\%$, `elfPlace` legalizes and fixes DSP and RAM blocks using the minimum-cost flow approach like in [8], [9]. Here we set a larger overflow target for DSP/RAM due to their much higher discreteness compared with LUT/FF. After that, LUT/FF placements are further optimized until they both meet the overflow target again ($\max(O_{\text{LUT}}, O_{\text{FF}}) < 10\%$). Finally, `elfPlace` adopts the clustering, legalization, and detailed placement approaches proposed in [16] to produce the final legal solution.

IV. Core Placement ALgorithms

A. The Augmented Lagrangian Formulation

With density constraint for each resource type modeled as a separate electrostatic system, `elfPlace` solves the minimization problem defined as follows.

$$\min_{\boldsymbol{x},\boldsymbol{y}} \widetilde{W}(\boldsymbol{x},\boldsymbol{y}) \quad \text{s.t. } \Phi_s(\boldsymbol{x},\boldsymbol{y}) = 0, \forall s \in \mathcal{S}. \tag{8}$$

However, unlike `ePlace` solves the density constrained placement problem using the multiplier method given in Eq. (6), `elfPlace` uses the augmented Lagrangian method (ALM), as shown in Eq. (9), instead.

$$\min_{\boldsymbol{x},\boldsymbol{y}} f(\boldsymbol{x},\boldsymbol{y}) = \widetilde{W}(\boldsymbol{x},\boldsymbol{y}) + \sum_{s\in\mathcal{S}} \lambda_s \Big(\Phi_s(\boldsymbol{x},\boldsymbol{y}) + \frac{c_s}{2} \Phi_s(\boldsymbol{x},\boldsymbol{y})^2 \Big). \tag{9}$$

Here λ_s and Φ_s are density multiplier and electric potential energy for each resource type $s \in \mathcal{S} = \{\text{LUT, FF, DSP, RAM}\}$, and c_s is a parameter to control the relative weight of the quadratic penalty term $\Phi_s(\boldsymbol{x},\boldsymbol{y})^2$. Slightly different from the typical ALM formulation where $\boldsymbol{\Phi}(\boldsymbol{x},\boldsymbol{y})^2$ has weight independent to $\boldsymbol{\lambda}$, the magnitude of $\boldsymbol{\Phi}(\boldsymbol{x},\boldsymbol{y})^2$ is also determined by $\boldsymbol{\lambda}$ in Eq. (9). This is to better control the overall effort on honoring the density constraints and make `elfPlace` less sensitive to the initial placement.

The ALM formulation in Eq. (9) can be viewed as a mixture of the multiplier method and the penalty method. The motivation is that, when the resource type s has high potential energy $\Phi_s(\boldsymbol{x},\boldsymbol{y})$, we want the penalty term $\frac{c_s}{2}\Phi_s(\boldsymbol{x},\boldsymbol{y})^2$ to dominate (i.e., $\frac{c_s}{2}\Phi_s(\boldsymbol{x},\boldsymbol{y})^2 \gg \Phi_s(\boldsymbol{x},\boldsymbol{y})$) and make Eq. (9) become the penalty method as shown in Eq. (10). Since in this case, the resource type s still has lots of overlaps, using the penalty method can enhance the convexity of the objective function and improve the convergence.

$$\min_{\boldsymbol{x},\boldsymbol{y}} f_{\text{PM}}(\boldsymbol{x},\boldsymbol{y}) = \widetilde{W}(\boldsymbol{x},\boldsymbol{y}) + \sum_{s\in\mathcal{S}} \lambda_s \frac{c_s}{2} \Phi_s(\boldsymbol{x},\boldsymbol{y})^2. \tag{10}$$

On the other hand, when the resource type s converges to a relatively small potential energy $\Phi_s(\boldsymbol{x},\boldsymbol{y})$, we want the $\Phi_s(\boldsymbol{x},\boldsymbol{y})$ term to dominate (i.e., $\Phi_s(\boldsymbol{x},\boldsymbol{y}) \gg \frac{c_s}{2}\Phi_s(\boldsymbol{x},\boldsymbol{y})^2$) and make Eq. (9) become the multiplier method as shown in Eq. (11). In this case, the overlaps of the resource type s are already relatively small and using the multiplier method can continue the optimization without suffering the ill-conditioning problem associated with the penalty method.

$$\min_{\boldsymbol{x},\boldsymbol{y}} f_{\text{MM}}(\boldsymbol{x},\boldsymbol{y}) = \widetilde{W}(\boldsymbol{x},\boldsymbol{y}) + \sum_{s\in\mathcal{S}} \lambda_s \Phi_s(\boldsymbol{x},\boldsymbol{y}). \tag{11}$$

The key of achieving this penalty method and multiplier method trade-off is to properly set the value of $c_s, \forall s \in \mathcal{S}$. We observe that, regardless of the design size, the final potential energy always converges to 10^{-5} to 10^{-7} of the initial one. Therefore, we define c_s as follows.

$$c_s = \frac{\beta}{\Phi_s(\boldsymbol{x}^{(0)},\boldsymbol{y}^{(0)})}, \forall s \in \mathcal{S}, \tag{12}$$

where β is set to 2×10^3 in our experiments and $\Phi_s(\boldsymbol{x}^{(0)},\boldsymbol{y}^{(0)})$ is the potential energy of the random initial placement (described in Section III). Under this setting, we will have $\frac{c_s}{2}\Phi_s(\boldsymbol{x},\boldsymbol{y})^2 = \Phi_s(\boldsymbol{x},\boldsymbol{y})$ when $\Phi_s(\boldsymbol{x},\boldsymbol{y}) = 10^{-3}\,\Phi_s(\boldsymbol{x}^{(0)},\boldsymbol{y}^{(0)})$. Then, the penalty method can smoothly transit to multiplier method at about the halfway of the final convergence. The experimental result in Section VI-B shows that our ALM formulation could improve the final routed wirelength by 1.2% compared with the original multiplier method adopted in `ePlace`.

B. Gradient Computation and Preconditioning

The x-directed gradient of our objective function defined in Eq. (9) can be derived as shown in Eq. (13). For brevity, only x-direction will be discussed in the rest of this section and similar conclusions are applicable to y-direction as well.

$$\begin{aligned}\frac{\partial f(\boldsymbol{x},\boldsymbol{y})}{\partial x_i} &= \frac{\partial \widetilde{W}(\boldsymbol{x},\boldsymbol{y})}{\partial x_i} + \lambda_s\Big(\frac{\partial \Phi_s(\boldsymbol{x},\boldsymbol{y})}{\partial x_i} + c_s\Phi_s(\boldsymbol{x},\boldsymbol{y})\frac{\partial \Phi_s(\boldsymbol{x},\boldsymbol{y})}{\partial x_i}\Big)\\ &= \frac{\partial \widetilde{W}(\boldsymbol{x},\boldsymbol{y})}{\partial x_i} - \lambda_s q_i \xi_{x_i}\Big(1 + c_s\Phi_s(\boldsymbol{x},\boldsymbol{y})\Big),\ \ \forall i \in \mathcal{V}_s.\end{aligned} \tag{13}$$

Although our ALM-based density penalty term is initially motivated by mathematics, there are still physical intuitions behind its gradient. By the nature of electrostatics, the electric force $q_i\boldsymbol{\xi}_i$ on each charge i will guide the charge towards a nearby low-potential well and this is reflected by the $\lambda_s q_i \boldsymbol{\xi}_{x_i}$ term in Eq. (13). Besides, the extra $\lambda_s q_i \boldsymbol{\xi}_{x_i} c_s \Phi_s(\boldsymbol{x},\boldsymbol{y})$ term further accelerates the charge movement for resource types with high potential energies, which often correspond to relatively large cell overlaps in the placement problem.

The gradient defined in Eq. (13) will be preconditioned before being finally fed to the optimizer. Preconditioning can make the local curvature of the objective function become nearly spherical, and hence, alleviate the ill-conditioning problem and improve the numerical convergence and stability. The most commonly used preconditioner is the inverse of the Hessian matrix $\boldsymbol{H}_f$ of the objective function f, and the preconditioned gradient $\boldsymbol{H}_f^{-1}\nabla f$, instead of the original ∇f, will be used as the (opposite of) descent direction. However, due to the scale of placement problem and the complexity of our objective function, it is impractical to compute the exact Hessian. Instead, `elfPlace` adopts the much cheaper Jacobi preconditioner to approximate the actual Hessian.

The x-directed Jacobi preconditioner is a diagonal matrix with the i-th diagonal entry equal to $\frac{\partial^2 f}{\partial x_i^2}$. By Eq. (13), we have

$$\frac{\partial^2 f(\boldsymbol{x},\boldsymbol{y})}{\partial x_i^2} = \frac{\partial^2 \widetilde{W}(\boldsymbol{x},\boldsymbol{y})}{\partial x_i^2} - \lambda_s q_i\Big(\frac{\partial \xi_{x_i}}{\partial x_i}\Big(1 + c_s\Phi(\boldsymbol{x},\boldsymbol{y})\Big) - c_s\xi_{x_i}^2\Big). \tag{14}$$

The closed-form expression of $\frac{\partial^2 \widetilde{W}(\boldsymbol{x},\boldsymbol{y})}{\partial x_i^2}$ is too expensive to compute in practice, therefore, we approximate it using

$$\frac{\partial^2 \widetilde{W}(\boldsymbol{x},\boldsymbol{y})}{\partial x_i^2} \ \sim\ \sum_{e\in\mathcal{E}_i}\frac{1}{|e|-1}, \tag{15}$$

where $\mathcal{E}_i$ denotes the set of nets incident to instance i and $|e|$ denotes the degree of the net e. The second-order derivative of the density term is even more complicated. Although the numerical solution of $\frac{\partial \xi_{x_i}}{\partial x_i}$ can be computed again through spectral method based on Eq. (5), we choose to only keep the $\lambda_s q_i$ term for the sake of efficiency.

Therefore, the overall x-directed second-order derivative of the objective is approximated as follows.

$$\frac{\partial^2 f(\boldsymbol{x},\boldsymbol{y})}{\partial x_i^2} \ \sim\ h_{x_i} = \max\Big(\sum_{e\in\mathcal{E}_i}\frac{1}{|e|-1} + \lambda_s q_i,\ 1\Big), \forall i \in \mathcal{V}_s, \tag{16}$$

where the $\max(\cdot, 1)$ is to avoid extremely small h_{x_i} for filler instances, who do not have incident nets, when λ_s is very small. Finally, the preconditioned gradient,

$$H_f^{-1}\nabla f(\boldsymbol{x},\boldsymbol{y}) = \Big(\frac{1}{h_{x_1}}\frac{\partial f(\boldsymbol{x},\boldsymbol{y})}{\partial x_1}, \frac{1}{h_{y_1}}\frac{\partial f(\boldsymbol{x},\boldsymbol{y})}{\partial y_1}, \cdots\Big)^T, \tag{17}$$

will be fed to a Nesterov's optimizer [24] to iteratively update the placement solution. Since in FPGA designs, net degrees (e.g., local signal nets and global clock nets) and instance pin counts and sizes (e.g., small LUT/FF instances and large DSP/RAM blocks) can vary significantly, this wirelength preconditioner is essential to the numerical convergence of our optimization. The experimental result in Section VI-B shows that `elfPlace` can barely converge without our preconditioning technique.

C. Density Multipliers Setting

One important thing we have not yet discussed is the setting of density multipliers $\boldsymbol{\lambda}$, which control the spreading efforts on different resource types. Since there is often heavy connectivity among different resource types, the spreading process must be capable of achieving target densities for all resource types while not ruining the natural physical clusters consisting of heterogeneous instances.

In `elfPlace`, we set the initial density multipliers $\boldsymbol{\lambda}^{(0)}$ as follows.

$$\boldsymbol{\lambda}^{(0)} = \eta\frac{\|\nabla\widetilde{W}(\boldsymbol{x}^{(0)},\boldsymbol{y}^{(0)})\|_1}{\sum_{i\in\mathcal{V}}\ q_i\|\boldsymbol{\xi}_i^{(0)}\|_1}\ (1,1,\cdots,1)^T, \tag{18}$$

where $(\boldsymbol{x}^{(0)},\boldsymbol{y}^{(0)})$ represent the initial placement, $\boldsymbol{\xi}_i^{(0)}$ denotes the initial electric field at instance i, η is a weighting parameter, and $\|\cdot\|_1$ denotes the L1-norm of a vector. In order to emphasize the wirelength optimization in early iterations, η is set to 10^{-4} in our experiments. Note that $\boldsymbol{\lambda}^{(0)}$ is an $|\mathcal{S}|$-dimensional vector, where $|\mathcal{S}|$ is the number of resource types, and by Eq. (18), we start from spreading all resource types with the same weight.

Classical optimization approaches use the subgradient method to update $\boldsymbol{\lambda}$ [25]. According to Eq. (9), the subgradient of $\boldsymbol{\lambda}$ is defined as

$$\nabla_{\text{sub}}\boldsymbol{\lambda} = \Big(\cdots, \Phi_s(\boldsymbol{x},\boldsymbol{y}) + \frac{c_s}{2}\Phi_s(\boldsymbol{x},\boldsymbol{y})^2, \cdots\Big)^T, s \in \mathcal{S}. \tag{19}$$

The reason why $\nabla_{\text{sub}}\boldsymbol{\lambda}$ is called subgradient instead of gradient is that the dual function, $l(\boldsymbol{\lambda}) = \max f(\boldsymbol{x},\boldsymbol{y})|_{\boldsymbol{\lambda}}$, associated with Eq. (9) is not smooth but piecewise linear [25].

However, in our placement problem, the potential energies of different resource types, Φ_s, can differ by order of magnitudes. The very sparse DSP/RAM blocks usually have significantly smaller total potential energies compared with LUT/FF instances. As a result, using the subgradient in Eq. (19) to guide the $\boldsymbol{\lambda}$ updating can lead to severely ill-conditioned problems. To mitigate this issue, the normalized subgradient defined in Eq. (20) is used instead in `elfPlace`.

$$\begin{aligned}\widehat{\nabla}_{\text{sub}}\boldsymbol{\lambda} &= \Big(\cdots, \frac{1}{\Phi_s(\boldsymbol{x}^{(0)},\boldsymbol{y}^{(0)})}\Big(\Phi_s(\boldsymbol{x},\boldsymbol{y}) + \frac{c_s}{2}\Phi_s(\boldsymbol{x},\boldsymbol{y})^2\Big), \cdots\Big)^T,\\ &= \Big(\cdots, \widehat{\Phi}_s(\boldsymbol{x},\boldsymbol{y}) + \frac{\beta}{2}\widehat{\Phi}_s(\boldsymbol{x},\boldsymbol{y})^2, \cdots\Big)^T, s \in \mathcal{S}.\end{aligned} \tag{20}$$

Here we use $\widehat{\Phi}_s(\boldsymbol{x},\boldsymbol{y}) = \Phi_s(\boldsymbol{x},\boldsymbol{y})/\Phi_s(\boldsymbol{x}^{(0)},\boldsymbol{y}^{(0)})$ to denote the potential energy normalized by the potential energy of the initial placement and c_s is replaced by its definition given in Eq. (12). After this normalization, each $\widehat{\Phi}_s(\boldsymbol{x},\boldsymbol{y})$ is approximately upper bounded by 1, which can more accurately reflect the relative level of density violation for each resource type.

Given the $\boldsymbol{\lambda}^{(k)}$ and the step size $t^{(k)}$ at iteration k, we compute $\boldsymbol{\lambda}^{(k+1)}$ by Eq. (21) and our step size updating scheme is further presented by Eq. (22).

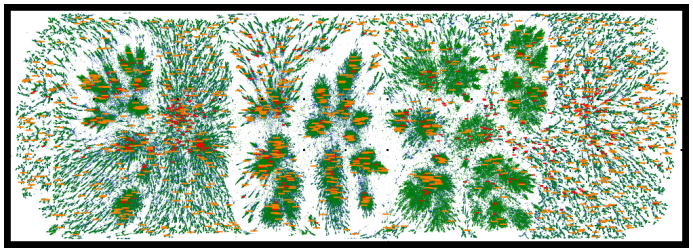
$$\boldsymbol{\lambda}^{(k+1)} = \boldsymbol{\lambda}^{(k)} + t^{(k)}\frac{\widehat{\nabla}_{\text{sub}}\boldsymbol{\lambda}^{(k)}}{\|\widehat{\nabla}_{\text{sub}}\boldsymbol{\lambda}^{(k)}\|_2}. \tag{21}$$

$$t^{(k)} = \begin{cases}\alpha_{\text{H}} - 1, & \text{for } k = 0,\\ t^{(k-1)}\Big(\frac{\log(\beta\|\widehat{\boldsymbol{\Phi}}^{(k)}\|_2+1)}{1+\log(\beta\|\widehat{\boldsymbol{\Phi}}^{(k)}\|_2+1)}(\alpha_{\text{H}} - \alpha_{\text{L}}) + \alpha_{\text{L}}\Big), & \text{for } k > 0.\end{cases} \tag{22}$$

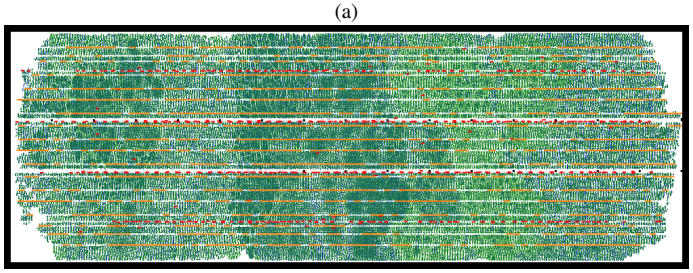
Here β is the same weighting parameter used in Eq. (12) and the parameter pair $(\alpha_{\text{L}}, \alpha_{\text{H}})$ defines the range of increasing rate of the

step size. The motivation of our step size updating scheme shown in Eq. (22) is that the quadratic density penalty term often decays much faster than the linear penalty term in our objective Eq. (9). Therefore, we incline to increase the step size faster when the quadratic penalty term dominates (i.e., $\beta\|\widehat{\mathbf{\Phi}}^{(k)}\|_2 \gg 1$). As the instance overlaps become smaller, the linear penalty term will start to take over and a slower increasing rate will be used in this case. In our experiments, we set $(\alpha_{\mathrm{L}}, \alpha_{\mathrm{H}})$ to $(1.05, 1.06)$. It should be noted that, although $1.05 \approx 1.06$, their high-order exponents can differ by order of magnitudes (e.g., $(1.06/1.05)^{500} > 100$).

Figure 3 illustrates the heterogeneous spreading process in `elfPlace`. As can be seen from Fig. 3(a), our $\boldsymbol{\lambda}$ updating scheme can greatly preserve those natural physical clusters consisting of heterogeneous instances. The placement right before DSP/RAM legalization shown in Fig. 3(b) further demonstrates the capability of `elfPlace` to achieve nearly overlap-free solutions even for highly-discrete DSP/RAM blocks without explicit legalization.



(a)



(b)

Fig. 3: The distributions of physical LUT (green), FF (blue), DSP (red), and RAM (orange) instances in (a) an intermediate placement and (b) the placement right before DSP/RAM legalization based on `FPGA-10`. Both figures are rotated by 90 degrees.

V. Instance Area Adjustment

Besides minimizing wirelength, `elfPlace` is also capable of tackling other practical issues in real-world designs, such as routability, pin density, and clustering compatibility. Routability and pin density optimizations have always been the fundamental requirements of placement to achieve routing-friendly solutions. While clustering compatibility optimization, which was recently discussed in [16], is to further consider the effect of downstream clustering (also referred as packing) early in the placement stage. It turns out that all these issues can be addressed by properly adjusting instance areas on top of our wirelength-driven placement. Therefore, in this section, we propose a unified instance area adjustment approach to simultaneously optimize all of them.

A. The Adjustment Scheme

Figure 4 sketches the algorithm flow of our instance area adjustment. In the beginning, for each physical instance i, we first compute three independent instance areas one each is optimized for routability (A_i^{ro}), pin density (A_i^{po}), and clustering compatibility (A_i^{co}). Let A_i denote

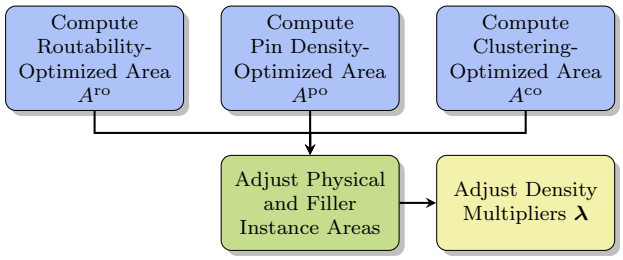


Fig. 4: The area adjustment flow in `elfPlace` to simultaneously optimize routability, pin density, and downstream clustering compatibility.

the area of instance i before the adjustment, we define the target area increase of each physical instance i as follows.

$$\Delta A_i = \max(A_i^{\mathrm{ro}}, A_i^{\mathrm{po}}, A_i^{\mathrm{co}}, A_i) - A_i, \forall i \in \mathcal{V}^{\mathrm{P}}. \tag{23}$$

In order to prevent the total adjusted area from exceeding the total capacity of each resource type, all ΔA_i need to be further scaled by the following factor according to the resource type s of i.

$$\tau_s = \min\left(\frac{\sum_{i\in\mathcal{V}_s^{\mathrm{F}}} A_i}{\sum_{i\in\mathcal{V}_s^{\mathrm{P}}} \Delta A_i}, 1\right), \forall s \in \mathcal{S}, \tag{24}$$

where $\mathcal{V}_s^{\mathrm{F}}$ denotes the set of filler instances for resource type s and $\sum_{i\in\mathcal{V}_s^{\mathrm{F}}} A_i$ is the total filler area of resource type s before the adjustment. Basically, scaling all ΔA_i by Eq. (24) guarantees that the total increased physical instance area is no greater than the total available filler area for each resource type. The final adjusted area A_i' of each physical instance i is then given by Eq. (25).

$$A_i' = A_i + \tau_s \Delta A_i, \forall i \in \mathcal{V}_s^{\mathrm{P}}, \forall s \in \mathcal{S}. \tag{25}$$

Recall that `elfPlace` relies on electrostatic neutrality to meet the density constraints $\mathbf{\Phi} = \mathbf{0}$, therefore, we also need to downsize filler instances to maintain the total positive charge quantity unchanged. The final adjusted area A_i' of each filler instance i is then defined as follows.

$$A_i' = \frac{\sum_{j\in\mathcal{V}_s} A_j - \sum_{j\in\mathcal{V}_s^{\mathrm{P}}} A_j'}{|\mathcal{V}_s^{\mathrm{F}}|}, \forall i \in \mathcal{V}_s^{\mathrm{F}}, \forall s \in \mathcal{S}. \tag{26}$$

Our area adjustment step is physically equivalent to redistributing charge density with the overall electrostatic neutrality preserved. After this redistribution, however, the previously nearly-equilibrated electrostatic states are likely to be ruined. In addition, the adjustment magnitudes and the potential energy increases can be highly uneven across different resource types (e.g., FFs can vary more than LUTs due to the control set rules). Therefore, we reset the density multipliers $\boldsymbol{\lambda}$ by Eq. (27) to adapt and recover from this perturbation.

$$\boldsymbol{\lambda}' = \eta' \frac{\|\nabla\widetilde{W}\|_1}{\langle(\cdots, \sum_{i\in\mathcal{V}_s} q_i\|\boldsymbol{\xi}_i\|_1, \cdots)^T, \widehat{\nabla}_{\mathrm{sub}}\boldsymbol{\lambda}\rangle} \widehat{\nabla}_{\mathrm{sub}}\boldsymbol{\lambda}, \tag{27}$$

where $\langle\cdot,\cdot\rangle$ denotes the inner product of two vectors, $(\cdots, \sum_{i\in\mathcal{V}_s} q_i\|\boldsymbol{\xi}_i\|_1, \cdots)^T$ is an $|\mathcal{S}|$-dimensional vector that contains the L1-norm of the density gradient for each resource type $s \in \mathcal{S}$, $\widehat{\nabla}_{\mathrm{sub}}\boldsymbol{\lambda}$ denotes the normalized subgradient of $\boldsymbol{\lambda}$, as defined in Eq. (20), after the area adjustment, and η' is a weighting parameter set to 0.1 in our experiments. Equation (27) essentially redirects $\boldsymbol{\lambda}$ to its current normalized subgradient $\widehat{\nabla}_{\mathrm{sub}}\boldsymbol{\lambda}$ with the scale determined by the gradient norm ratio between wirelength and density. In this way, both the direction and scale of the adjusted $\boldsymbol{\lambda}'$ can adapt the perturbed electrostatic system and help to better heal the placement quality. Besides, in order to smooth the placement convergence, the

density multiplier step size is also adjusted by Eq. (28), where α_{H} is the same parameter as used in Eq. (22).

$$t' = (\alpha_{\mathrm{H}} - 1)\,\|\boldsymbol{\lambda}'\|_2. \tag{28}$$

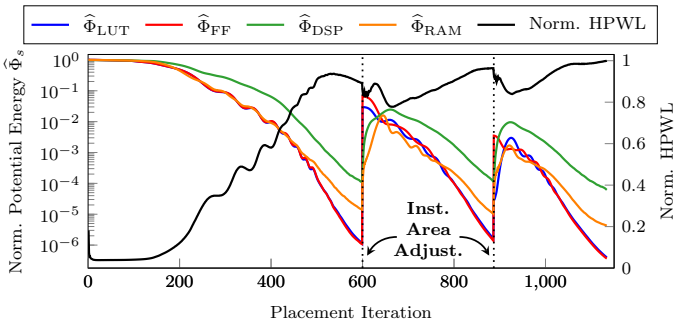


Fig. 5: The normalized potential energy $\widehat{\boldsymbol{\Phi}}$ and HPWL at different placement iterations on FPGA-10.

Figure 5 illustrates the impact of instance area adjustment on the placement convergence process. In this example, the adjustment is performed twice at iteration 600 and 887, where the potential energies increase sharply. By using our adaptive density multiplier and step size resetting techniques, the wirelength can be gradually healed and smoothly converges to a nearly overlap-free solution.

B. The Optimized Area Computation

As we discussed in Section V-A, for each physical instance i, elfPlace computes three independent areas that are optimized for routability (A_i^{ro}), pin density (A_i^{po}), and clustering compatibility (A_i^{co}), respectively.

1) The Routability-Optimized Area: In order to compute the routability-optimized areas, elfPlace first performs a RISA/RUDY-based [26], [27] routing congestion estimation. Let u_i^{h} and u_i^{v} denote the resulting horizontal and vertical routing utilizations at instance i, then we compute the routability-optimized area of each physical instance i using Eq. (29), where the 2 is an empirical constant to avoid overinflation.

$$A_i^{\mathrm{ro}} = A_i \min\Big(\max\big(u_i^{\mathrm{h}}, u_i^{\mathrm{v}}\big)^2, 2\Big), \forall i \in \mathcal{V}^{\mathrm{P}}, \tag{29}$$

2) The Pin Density-Optimized Area: Similarly, elfPlace also estimates pin density by dividing the placement region into bins. Let c^{p} denote the unit-area pin capacity (determined by the FPGA architecture). For each instance i, if we denote its local pin density by u_i^{p} and denote its pin count as $|\mathcal{P}_i|$, then its pin density-optimized area is defined by Eq. (30), where the 1.5 is an empirical constant to avoid overinflation.

$$A_i^{\mathrm{po}} = \frac{|\mathcal{P}_i|}{c^{\mathrm{p}}} \min\big(u_i^{\mathrm{p}}, 1.5\big), \forall i \in \mathcal{V}^{\mathrm{P}}. \tag{30}$$

Different from the routability-optimized area A_i^{ro}, the pin density-optimized area A_i^{po} here is independent to the current instance area A_i. This is because, compared with routing utilization, local pin density is usually very noisy and sensitive to the placement. If A_i^{po} is self-accumulated as in Eq. (29), it can be excessively over-inflated.

3) The Clustering Compatibility-Optimized Area: One special challenge of flat FPGA placement is that we can barely know the correct LUT and FF areas before the actual downstream clustering solution is formed. Recall the CLB architecture described in Section II-A, if a LUT/FF is incompatible with most of its physical neighbors (e.g., violating the pin count and control set rules), then it tends to occupy a significant portion of a CLB alone. For such an instance, we intuitively should assign it a larger area.

To estimate the instance areas in a feasible clustering solution, we first assume the instance movement $(\Delta\boldsymbol{x}, \Delta\boldsymbol{y})$ during the downstream clustering/legalization approximately follows Gaussian distribution. That is, we have $\Delta x_i \sim \mathcal{N}(0, \sigma)$ and $\Delta y_i \sim \mathcal{N}(0, \sigma)$, where σ is the assumed standard deviation of the movement. We empirically set σ to $\sqrt{10^{-5} \times |\mathcal{V}^{\mathrm{P}}|}$. Then, we divide the placement region into square bins with bin length equal to σ, and for each LUT/FF instance i, we conduct the area estimation using the bin window $\mathcal{B}_i$, of size 5×5 bins, that is centered at (x_i, y_i). Let $(\mathcal{B}_i^{\mathrm{xl}}, \mathcal{B}_i^{\mathrm{yl}}, \mathcal{B}_i^{\mathrm{xh}}, \mathcal{B}_i^{\mathrm{yh}})$ denote the bounding box of the estimation window $\mathcal{B}_i$ of instance i, the expectation of any instance j falling into $\mathcal{B}_i$ then can be defined as

$$\mathbf{E}_{j\in\mathcal{B}_i} = P_\sigma(\mathcal{B}_i^{\mathrm{xl}} \le x_j < \mathcal{B}_i^{\mathrm{xh}})\; P_\sigma(\mathcal{B}_i^{\mathrm{yl}} \le y_j < \mathcal{B}_i^{\mathrm{yh}}), \tag{31}$$

where $P_\sigma(a \le \mu < b)$ represents the total probability of the Gaussian distribution $\mathcal{N}(\mu, \sigma)$ in the range $[a, b)$.

For each LUT instance i, let $\mathcal{V}_i^{\mathrm{P}}$ and $\overline{\mathcal{V}_i^{\mathrm{P}}}$ denote the sets of LUTs that can and cannot be fitted into the same BLE with i (see Section II-A), respectively, then we define the clustering compatibility-optimized area for LUT i as follows.

$$A_i^{\mathrm{co}} = \frac{1}{16}\frac{\sum_{j\in\mathcal{V}_i^{\mathrm{P}}} \mathbf{E}_{j\in\mathcal{B}_i}}{\sum_{j\in\mathcal{V}_{\mathrm{LUT}}^{\mathrm{P}}} \mathbf{E}_{j\in\mathcal{B}_i}} + \frac{1}{8}\frac{\sum_{j\in\overline{\mathcal{V}_i^{\mathrm{P}}}} \mathbf{E}_{j\in\mathcal{B}_i}}{\sum_{j\in\mathcal{V}_{\mathrm{LUT}}^{\mathrm{P}}} \mathbf{E}_{j\in\mathcal{B}_i}}, \forall i \in \mathcal{V}_{\mathrm{LUT}}^{\mathrm{P}}. \tag{32}$$

Equation (32) essentially is a weighted average of the compatible and incompatible expectations for i within the window $\mathcal{B}_i$. Each $A_i^{\mathrm{co}}, \forall i \in \mathcal{V}_{\mathrm{LUT}}^{\mathrm{P}}$, is in the range $[1/16, 1/8]$ based on our target architecture. The same idea is also adopted in [16] and the proposed Equation (32) is its enhancement with Gaussian smoothing.

The estimation for FFs are more subtle due to the complicated control set rules (see Section II-A). For an FF instance i, let θ_i denote its control set $(\mathrm{CK}, \mathrm{SR}, \mathrm{CE})$ and let Θ_i denote the set of control sets that have the same CK and SR with θ_i. If we use $n_{i,\theta}$ to denote the number of FFs in $\mathcal{B}_i$ with the control set θ, then the area of FF i in the tightest clustering solution formed within $\mathcal{B}_i$ can be estimated by Eq. (33), as given in [16].

$$A_i^{\text{co-disc}} = \frac{1}{2n_{i,\theta_i}}\frac{\lceil n_{i,\theta_i}/4\rceil}{\sum_{\theta\in\Theta_i}\lceil n_{i,\theta}/4\rceil}\left\lceil\frac{\sum_{\theta\in\Theta_i}\lceil n_{i,\theta}/4\rceil}{2}\right\rceil, \forall i \in \mathcal{V}_{\mathrm{FF}}^{\mathrm{P}}. \tag{33}$$

We omit the derivation of Eq. (33) due to the page limit. However, it still can be seen that Eq. (33) involves many ceiling operations ($\lceil\cdot\rceil$), which make $A_i^{\text{co-disc}}$ discontinuous (disc) and very sensitive to the estimation window $\mathcal{B}_i$ and the placement solution.

In elfPlace, the much smoother Eq. (34), instead of Eq. (33), is used as the clustering compatibility-optimized areas for FF instances.

$$A_i^{\mathrm{co}} = \frac{1}{2\mathbf{E}_{i,\theta_i}}\frac{\mathrm{sdc}(\mathbf{E}_{i,\theta_i}, 4)}{\sum_{\theta\in\Theta_i}\mathrm{sdc}(\mathbf{E}_{i,\theta}, 4)}\mathrm{sdc}\Big(\sum_{\theta\in\Theta_i}\mathrm{sdc}(\mathbf{E}_{i,\theta}, 4),\ 2\Big), \forall i \in \mathcal{V}_{\mathrm{FF}}^{\mathrm{P}}. \tag{34}$$

It has two notable improvements over Eq. (33): (1) it replaces each FF count $n_{i,\theta}$ in Eq. (33) with the smoother expectation $\mathbf{E}_{i,\theta}$, which denotes the expected number (nonintegral in general) of FFs in window $\mathcal{B}_i$ with the control set θ; (2) it replaces each division-ceiling operation in Eq. (33) with the soft division-ceiling function $\mathrm{sdc}(x, d)$ defined in Eq. (35).

$$\mathrm{sdc}(x, d) = \begin{cases} x + (1-d)\lfloor x/d\rfloor, & \text{for } x/d - \lfloor x/d\rfloor < 1/d, \\ \lceil x/d\rceil, & \text{otherwise.} \end{cases} \tag{35}$$

The plots of $\mathrm{sdc}(x, d)$ function w.r.t. x/d are illustrated in Fig. 6. It smoothes $\lceil x/d\rceil$ by linearizing the beginning $1/d$ of each sharp step. As d approaches to ∞, $\mathrm{sdc}(x, d)$ behaves more like $\lceil x/d\rceil$.

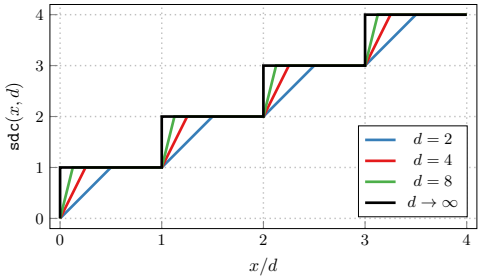


Fig. 6: The plots of the soft division-ceiling function $\texttt{sdc}(x, d)$ w.r.t. x/d.

VI. Experimental Results

We implement elfPlace in C++ and perform experiments on a Linux machine running with Intel Core i9-7900 CPUs (3.30 GHz and 10 cores) and 128 GB RAM. Careful parallelization is applied throughout the whole framework with the support of OpenMP 4.0 [28]. The ISPD 2016 FPGA placement contest benchmark suite [19] released by Xilinx is adopted to demonstrate the effectiveness and efficiency of elfPlace [1]. Routed wirelength reported by Xilinx Vivado v2015.4 is used to evaluate the placement quality. The characteristics of the benchmarks are listed in Table III.

TABLE III: ISPD 2016 Contest Benchmarks Statistics

Design	#LUT	#FF	#RAM	#DSP	#Ctrl Set
FPGA-01	50K	55K	0	0	12
FPGA-02	100K	66K	100	100	121
FPGA-03	250K	170K	600	500	1281
FPGA-04	250K	172K	600	500	1281
FPGA-05	250K	174K	600	500	1281
FPGA-06	350K	352K	1000	600	2541
FPGA-07	350K	355K	1000	600	2541
FPGA-08	500K	216K	600	500	1281
FPGA-09	500K	366K	1000	600	2541
FPGA-10	350K	600K	1000	600	2541
FPGA-11	480K	363K	1000	400	2091
FPGA-12	500K	602K	600	500	1281
Resources	538K	1075K	1728	768	-

A. Comparison with State-of-the-Art Placers

We compare elfPlace with four state-of-the-art analytical FPGA placers, namely, UTPlaceF [8], RippleFPGA [9], GPlace3.0 [11], and UTPlaceF-DL [16]. The executables are obtained from their authors and executed on our machine. Since only UTPlaceF-DL and elfPlace support multi-threading, UTPlaceF, RippleFPGA, and GPlace3.0 are single-thread executed, while UTPlaceF-DL and elfPlace are executed with both a single thread and 10 threads.

Table IV shows the comparison results. Metrics "WL" and "RT" represent the routed wirelength in thousands and runtime in seconds, while "WLR" and "RTR" represent the routed wirelength and runtime ratios normalized to the 10-threaded elfPlace. It can be seen that elfPlace achieves the best routed wirelength on eleven out of twelve designs and outperforms UTPlaceF, RippleFPGA, GPlace3.0, and UTPlaceF-DL by, on average, 13.6%, 11.3%, 8.9%, and 7.1%, respectively. It is worthwhile to note that these wirelength improvements are fairly consistent from small designs to large ones. With only a single thread, elfPlace demonstrates similar runtime compared with UTPlaceF, GPlace3.0, and UTPlaceF-DL, while it runs more than 3× slower than the fastest RippleFPGA. By exploiting 10 threads, elfPlace achieves 3.51× speedup and shows similar runtime with RippleFPGA. Among all twelve designs, the 10-threaded elfPlace produces the fastest runtime on the seven largest designs, which evidences its good scalability.

[1]As this work focuses on core placement algorithms, the ISPD 2017 benchmark suite for clock-aware placement is not adopted in our experiments.

B. Individual Technique Validation

Table V further validates the effectiveness of each proposed technique. The column "w/ MM in Eq. (11)" shows the results of using the multiplier method (MM) in Eq. (11), which is adopted by ePlace, instead of our proposed augmented Lagrangian method (ALM) in Eq. (9). To make a fair comparison, we set the step size of the MM in a way that the MM and our ALM can converge within about the same amount of time. With this setup, our proposed ALM-based formulation can produce an average of 1.2% better routed wirelength compared with the MM-based formulation. The column "w/o Precond." shows the results without the preconditioning in Eq. (16) and it can barely converge due to the wide spectrum of instance sizes and net degrees in FPGA designs. The column "w/ ePlace Precond." further gives the results of replacing our preconditioner in Eq. (16) with the one proposed in ePlace [24]. Although the ePlace's preconditioning technique can achieve similar placement quality and efficiency, it fails to converge on two benchmarks in our experiments. Finally, the column "w/ A_i^{co} in [16]" presents the results of using the clustering compatibility-optimized area proposed in [16] instead of our Gaussian and sdc-smoothed Eq. (32) and Eq. (34). With our smoothing techniques, elfPlace can converge 15% faster while maintaining essentially the same solution quality.

C. Runtime Breakdown

Figure 7 shows the runtime breakdown of the 10-threaded elfPlace based on FPGA-12. The most time-consuming part is to compute the wirelength gradient $\nabla\widetilde{W}$, which takes 34.4% of the total runtime. The density gradient computation is relatively efficient and it consumes 7.5% of the total runtime on constructing the density maps $\boldsymbol{\rho}$ and 1.4% of the total runtime on computing the electric potential $\boldsymbol{\psi}$ and electric field $\boldsymbol{\xi}$ by Eq. (5). The parameter updating, which involves the computation of wirelength, overflow, potential energy $\boldsymbol{\Phi}$, etc., takes 7.3% of the total runtime. The clustering, legalization, and detailed placement algorithms adopted from [16] consumes total 37.5% of the runtime. While the remaining 11.9% of the runtime is spent on parsing, placement initialization, and the rest of runtime-insignificant tasks.

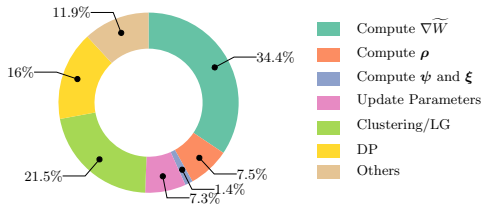


Fig. 7: The runtime breakdown of the 10-threaded elfPlace based on FPGA-12.

VII. Conclusion

In this paper, we have presented elfPlace, a general, flat, nonlinear placement algorithm for large-scale heterogeneous FPGAs. elfPlace resolves the traditional FPGA heterogeneity issue by casting the density constraints of heterogeneous resource types to separate but unified electrostatic systems. An augmented Lagrangian formulation together with a preconditioning technique and a normalized

TABLE IV: Routed Wirelength (WL in 10^3) and Placement Runtime (RT in seconds) Comparison with Other State-of-the-Art Placers

Design	UTPlaceF [8]				RippleFPGA [9]				GPlace3.0 [11]				UTPlaceF-DL [16]						elfPlace					
	WL	WLR	1-thread RT	1-thread RTR	WL	WLR	1-thread RT	1-thread RTR	WL	WLR	1-thread RT	1-thread RTR	WL	WLR	1-thread RT	1-thread RTR	10-thread RT	10-thread RTR	WL	WLR	1-thread RT	1-thread RTR	10-thread RT	10-thread RTR
FPGA-01	357	1.128	144	2.25	353	1.115	**25**	**0.39**	356	1.125	70	1.09	340	1.076	154	2.41	50	0.78	**316**	**1.000**	226	3.53	64	1.00
FPGA-02	642	1.106	244	2.02	645	1.112	**47**	**0.39**	644	1.110	133	1.10	653	1.125	273	2.26	90	0.74	**580**	**1.000**	429	3.55	121	1.00
FPGA-03	3215	1.123	672	3.26	3262	1.140	**168**	**0.82**	3101	1.084	502	2.44	3139	1.097	700	3.40	248	1.20	**2862**	**1.000**	732	3.55	206	1.00
FPGA-04	5410	1.117	667	3.06	5510	1.137	**188**	**0.86**	5403	1.115	537	2.46	5331	1.101	802	3.68	278	1.28	**4844**	**1.000**	781	3.58	218	1.00
FPGA-05	9660	1.048	841	3.61	9969	1.082	**229**	**0.98**	10507	1.140	639	2.74	10045	1.090	889	3.82	302	1.30	**9215**	**1.000**	845	3.63	233	1.00
FPGA-06	6488	1.133	1387	3.91	6180	1.079	356	1.00	5820	1.016	1147	3.23	5801	1.013	1128	3.18	424	1.19	**5727**	**1.000**	1124	3.17	**355**	**1.00**
FPGA-07	10105	1.155	1438	4.27	9640	1.102	394	1.17	9509	1.087	1428	4.24	9356	1.069	1277	3.79	479	1.42	**8749**	**1.000**	1092	3.24	**337**	**1.00**
FPGA-08	7879	1.028	1419	4.62	8157	1.065	354	1.15	8126	1.061	1575	5.13	8298	1.083	1478	4.81	497	1.62	**7661**	**1.000**	1214	3.95	**307**	**1.00**
FPGA-09	12369	1.161	2043	5.20	12305	1.155	485	1.23	11711	1.099	1938	4.93	11633	1.092	1724	4.39	622	1.58	**10657**	**1.000**	1359	3.46	**393**	**1.00**
FPGA-10	8795	1.452	2526	5.54	7140	1.178	547	1.20	6836	1.128	1797	3.94	6317	1.043	1467	3.22	583	1.28	**6058**	**1.000**	1445	3.17	**456**	**1.00**
FPGA-11	**10196**	**0.978**	1719	4.70	11023	1.058	447	1.22	10260	0.985	1786	4.88	10476	1.005	1687	4.61	640	1.75	10421	1.000	1367	3.73	**366**	**1.00**
FPGA-12	7755	1.197	2455	5.18	7363	1.136	549	1.16	7224	1.115	2296	4.84	6835	1.055	1926	4.06	771	1.63	**6480**	**1.000**	1714	3.62	**474**	**1.00**
Norm.	-	1.136	-	3.97	-	1.113	-	**0.96**	-	1.089	-	3.42	-	1.071	-	3.63	-	1.31	-	**1.000**	-	3.51	-	1.00

TABLE V: Normalized Routed Wirelength and Placement Runtime Comparison for Individual Technique Validation

Design	w/ MM in Eq. (11)		w/o Precond.		w/ ePlace Precond.		w/ A_i^{co} in [16]		elfPlace	
	WLR	RTR	WLR	RTR	WLR	RTR	WLR	RTR	WLR	RTR
FPGA-01	1.021	1.02	1.009	1.01	1.006	1.02	1.004	1.12	1.000	1.00
FPGA-02	1.010	0.97	*	*	*	*	1.001	1.16	1.000	1.00
FPGA-03	1.009	1.03	*	*	0.995	1.03	0.998	1.23	1.000	1.00
FPGA-04	1.046	0.94	*	*	*	*	1.002	1.16	1.000	1.00
FPGA-05	1.026	1.05	*	*	1.002	1.03	0.996	1.07	1.000	1.00
FPGA-06	1.008	1.01	*	*	1.030	0.98	1.004	1.25	1.000	1.00
FPGA-07	1.001	1.04	*	*	0.988	1.04	0.986	1.20	1.000	1.00
FPGA-08	0.991	1.01	*	*	0.996	1.01	0.999	1.14	1.000	1.00
FPGA-09	1.003	1.01	*	*	0.996	1.03	1.002	1.12	1.000	1.00
FPGA-10	1.006	1.01	*	*	1.000	1.01	0.996	1.03	1.000	1.00
FPGA-11	1.011	0.98	*	*	1.002	1.05	1.009	1.15	1.000	1.00
FPGA-12	1.017	1.02	*	*	0.995	1.04	1.003	1.14	1.000	1.00
Norm.	1.012	1.01	1.009	1.01	1.001	1.03	1.000	1.15	1.000	1.00

* Placement fails to converge.

subgradient-based multiplier updating scheme are proposed to achieve satisfiable solution quality with fast and robust numerical convergence. Besides pure-wirelength minimization, elfPlace is also capable of optimizing routability, pin density, and downstream clustering compatibility based on a unified instance area adjustment scheme. Our experiments show that elfPlace significantly outperforms four state-of-the-art placers in routed wirelength with competitive runtime. In the future, we plan to incorporate timing optimization into elfPlace framework.

ACKNOWLEDGMENT

This work was supported in part by Xilinx Inc. The authors would like to thank Dr. Gengjie Chen and Prof. Evangeline F.Y. Young for providing the binary of RippleFPGA and Dr. Ziad Abuowaimer, Prof. Shawki Areibi, and Prof. Gary Grewal for providing the binary of GPlace3.0.

REFERENCES

[1] V. Betz and J. Rose, "VPR: A new packing, placement and routing tool for FPGA research," in *FPL*, 1997, pp. 213–222.
[2] G. Chen and J. Cong, "Simultaneous placement with clustering and duplication," *ACM TODAES*, vol. 11, no. 3, pp. 740–772, 2006.
[3] P. Maidee, C. Ababei, and K. Bazargan, "Timing-driven partitioning-based placement for island style FPGAs," *IEEE TCAD*, vol. 24, no. 3, pp. 395–406, 2005.
[4] Y. Xu and M. A. Khalid, "QPF: efficient quadratic placement for FPGAs," in *FPL*, 2005, pp. 555–558.
[5] P. Gopalakrishnan, X. Li, and L. Pileggi, "Architecture-aware FPGA placement using metric embedding," in *DAC*, 2006, pp. 460–465.
[6] M. Xu, G. Gréwal, and S. Areibi, "StarPlace: A new analytic method for FPGA placement," *Integration, the VLSI Journal*, vol. 44, no. 3, pp. 192–204, 2011.
[7] M. Gort and J. H. Anderson, "Analytical placement for heterogeneous FPGAs," in *FPL*, 2012, pp. 143–150.
[8] W. Li, S. Dhar, and D. Z. Pan, "UTPlaceF: A routability-driven FPGA placer with physical and congestion aware packing," *IEEE TCAD*, vol. 37, no. 4, pp. 869–882, 2018.
[9] G. Chen, C.-W. Pui, W.-K. Chow, K.-C. Lam, J. Kuang, E. F. Young, and B. Yu, "RippleFPGA: Routability-driven simultaneous packing and placement for modern FPGAs," *IEEE TCAD*, vol. 37, no. 10, pp. 2022–2035, 2018.
[10] W. Li, Y. Lin, M. Li, S. Dhar, and D. Z. Pan, "UTPlaceF 2.0: A high-performance clock-aware FPGA placement engine," *ACM TODAES*, vol. 23, no. 4, p. 42, 2018.
[11] Z. Abuowaimer, D. Maarouf, T. Martin, J. Foxcroft, G. Gréwal, S. Areibi, and A. Vannelli, "GPlace3.0: Routability-driven analytic placer for UltraScale FPGA architectures," *ACM TODAES*, vol. 23, no. 5, pp. 66:1–66:33, 2018.
[12] T.-H. Lin, P. Banerjee, and Y.-W. Chang, "An efficient and effective analytical placer for FPGAs," in *DAC*, 2013, pp. 10:1–10:6.
[13] Y.-C. Chen, S.-Y. Chen, and Y.-W. Chang, "Efficient and effective packing and analytical placement for large-scale heterogeneous FPGAs," in *ICCAD*, 2014, pp. 647–654.
[14] Y.-C. Kuo, C.-C. Huang, S.-C. Chen, C.-H. Chiang, Y.-W. Chang, and S.-Y. Kuo, "Clock-aware placement for large-scale heterogeneous FPGAs," in *ICCAD*, 2017, pp. 519–526.
[15] N. K. Darav, A. Kennings, K. Vorwerk, and A. Kundu, "Multi-commodity flow-based spreading in a commercial analytic placer," in *FPGA*, 2019, pp. 122–131.
[16] W. Li and D. Z. Pan, "A new paradigm for FPGA placement without explicit packing," *IEEE TCAD*, 2018.
[17] J. Lu, H. Zhuang, P. Chen, H. Chang, C.-C. Chang, Y.-C. Wong, L. Sha, D. Huang, Y. Luo, C.-C. Teng *et al.*, "ePlace-MS: Electrostatics-based placement for mixed-size circuits," *IEEE TCAD*, vol. 34, no. 5, pp. 685–698, 2015.
[18] C.-K. Cheng, A. B. Kahng, I. Kang, and L. Wang, "RePlAce: Advancing solution quality and routability validation in global placement," *IEEE TCAD*, 2018.
[19] S. Yang, A. Gayasen, C. Mulpuri, S. Reddy, and R. Aggarwal, "Routability-driven FPGA placement contest," in *ISPD*, 2016, pp. 139–143.
[20] Xilinx Inc., "http://www.xilinx.com," 2019.
[21] M.-K. Hsu, Y.-W. Chang, and V. Balabanov, "TSV-aware analytical placement for 3D IC designs," in *DAC*, 2011, pp. 664–669.
[22] M.-K. Hsu, V. Balabanov, and Y.-W. Chang, "Tsv-aware analytical placement for 3-d ic designs based on a novel weighted-average wirelength model," *IEEE TCAD*, vol. 32, no. 4, pp. 497–509, 2013.
[23] Y. Lin, S. Dhar, W. Li, H. Ren, B. Khailany, and D. Z. Pan, "DREAMPlace: Deep learning toolkit-enabled gpu acceleration for modern VLSI placement," in *DAC*, 2019.
[24] 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 TODAES*, vol. 20, no. 2, p. 17, 2015.
[25] C. Lemaréchal, "Lagrangian relaxation," in *Computational combinatorial optimization*. Springer, 2001, pp. 112–156.
[26] C.-L. E. Cheng, "RISA: Accurate and efficient placement routability modeling," in *ICCAD*, 1994, pp. 690–695.
[27] P. Spindler and F. M. Johannes, "Fast and accurate routing demand estimation for efficient routability-driven placement," in *DATE*, 2007, pp. 1226–1231.
[28] OpenMP 4.0, http://www.openmp.org/, 2019, accessed: 2019-4-1.