# LEGALM: Efficient Legalization for Mixed-Cell-Height Circuits with Linearized Augmented Lagrangian Method


Jing Mai
jingmai@pku.edu.cn
School of CS, Peking University
Beijing, China

Chunyuan Zhao
zhaochunyuan@stu.pku.edu.cn
School of IC, Peking University
Beijing, China

Zuodong Zhang
zuodongzhang@pku.edu.cn
School of IC & Institute of EDA, Peking University
Beijing, China

Zhixiong Di
zxdi@home.swjtu.edu.cn
Southwest Jiaotong University
Chengdu, Sichuan, China

Yibo Lin*
yibolin@pku.edu.cn
School of IC & Institute of EDA, Peking University
Beijing Advanced Innovation Center for Integrated Circuits
Beijing, China

Runsheng Wang
Ru Huang
School of IC & Institute of EDA, Peking University
Beijing Advanced Innovation Center for Integrated Circuits
Beijing, China



## Abstract

Advanced technologies increasingly adopt mixed-cell-height circuits due to their superior power efficiency, compact area usage, enhanced routability, and improved performance. However, the complex constraints of modern circuit design, including routing challenges and fence region constraints, increase the difficulty of mixed-cell-height legalization. In this paper, we introduce **LEGALM**, a state-of-the-art mixed-cell-height legalizer that can address routability and fence region constraints more efficiently. We propose an augmented Lagrangian formulation coupled with a block gradient descent method that offers a novel analytical perspective on the mixed-cell-height legalization problem. To further enhance efficiency, we develop a series of GPU-accelerated kernels and a triple-fold partitioning technique with minor quality overhead. Experimental results on `ICCAD-2017` and modified `ISPD-2015` benchmarks show that our approach significantly outperforms current state-of-the-art legalization algorithms in both quality and efficiency.



## CCS Concepts

• **Hardware** → **Electronic design automation**; **Physical design (EDA)**; **Placement**; • **Computing methodologies** → **Parallel computing methodologies**; **Parallel algorithms**;

## Keywords

Legalization; Mixed-cell-height placement; Augmented Lagrangian optimization; Physical design; GPU acceleration

**ACM Reference Format:**
Jing Mai, Chunyuan Zhao, Zuodong Zhang, Zhixiong Di, Yibo Lin, Runsheng Wang, and Ru Huang. 2025. `LEGALM`: Efficient Legalization for Mixed-Cell-Height Circuits with Linearized Augmented Lagrangian Method . In *Proceedings of the 2025 International Symposium on Physical Design (ISPD '25), March 16–19, 2025, Austin, TX, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3698364.3705356

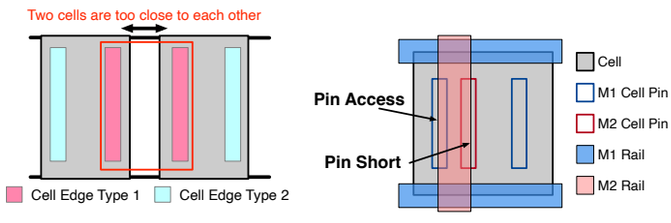


*Corresponding author.



*ISPD '25, Austin, TX, USA*






(a) (b)

**Figure 1: Illustrations for (a) edge spacing constraints and (b) pin-short/access constraints.**

## 1 Introduction

Legalization is a critical step in the modern VLSI physical design flow. This process involves refining global placement results by eliminating design rule violations while minimizing disruption to the global placement [1]. The quality of legalization significantly impacts the performance of subsequent stages, such as detailed placement and routing. Meanwhile, as global placement achieves substantial performance enhancements through GPU acceleration, legalization consequently emerges as one of the most time-consuming phases in the entire placement process [19]. Therefore, there is an urgent need for efficient and high-quality legalization algorithms to expedite design closure and maintain the pace of technological advancement in chip design.

Legalization for mixed-cell-height circuits emerges as a critical challenge in advanced process node design [12]. Compared to their single-row-height counterparts, multi-row-height cells provide enhanced drive capability and improved pin accessibility. Thus, multi-row-height cells are increasingly prevalent in cutting-edge technology nodes and play a pivotal role in achieving superior performance metrics. This trend underscores the growing importance of efficient mixed-cell-height legalization algorithms. Mixed-cell-height legalization faces more challenges, including more complex standard cell shapes and more intricate design rule constraints such as routability constraints and fence region constraints [12]. These factors result in more discrete mathematical problems, potentially lowering the solution quality of legalization algorithms.

Recently, the mixed-cell-height standard cell legalization problem has been studied extensively [4, 6–8, 10, 11, 13, 15–18, 22, 23, 26]. Existing literature can be categorized into two types based on whether their core algorithms allow cross-row movement: intra-row algorithms and inter-row algorithms.

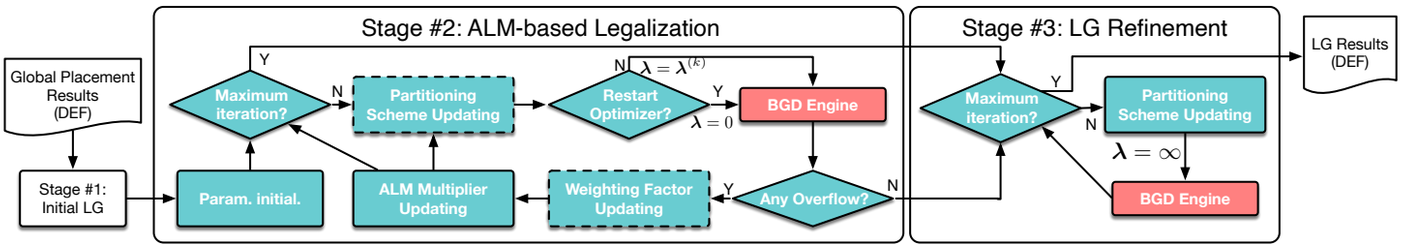



**Figure 2: The overflow for LEGALM algorithm. Dashed boxes indicate operations that may or may not be executed depending on whether certain conditions are met (see Algo. 1).**

The core algorithms of intra-row algorithms typically consist of two parts: row assignments and intra-row legalization algorithms based on the row assignment results. For row assignments, insertion point-based algorithms [7, 11, 17, 23] transform the row assignment task into a search for insertion points based on a greedy approach. They achieve better row assignment results by calculating more accurate displacement curves for the initial position and target location. For intra-row legalization algorithms, Abacus-like algorithms [22] improve upon the single-row-height legalization algorithm Abacus [21] and Tetris [14], addressing their insufficiencies and extending their advantages to resolve mixed-cell-height legalization. ILP-based algorithms [17] present an ILP model to minimize total displacement given the row assignment and intra-row order of cells, which is then transformed into a network flow problem for solving. ICP-based algorithms [7, 8, 18, 26] formulate the intra-row legalization problem under the row assignment and intra-row order of cells as a quadratic program and reformulate it into a linear complementary problem (LCP). The problem is then efficiently solved using a modulus-based matrix splitting iteration method (MMSIM), achieving excellent solution quality. However, these methods lack exploration of vertical movement and intra-row cell order, limiting the solution space.

The core algorithmic models of inter-row algorithms, on the other hand, allow cells to move across different rows. Among them, [4, 10, 13, 16] model the legalization problem as a network flow problem to quickly remove overlaps among the cells while minimizing cell displacement in both horizontal and vertical directions. [6] transforms the legalization problem into resource allocation tasks and proposes a negotiation-based legalization algorithm. [15] models the legalization problem as a cell spreading process and formulates it as an ILP problem. This method achieves high-quality solutions but requires considerable time compared with other legalization algorithms. Inter-row algorithms have more flexibility in solving mixed-height standard cell legalization problems, potentially utilizing more computational resources or employing heuristic methods to reduce unnecessary searches.

In this work, we propose LEGALM, a mixed-cell-height legalization algorithm that addresses routability and fence region constraints more efficiently. Our method provides a novel global perspective on the legalization problem from a mathematical programming standpoint. Our legalization methodology can significantly improve result quality and efficiency through better gradient information and GPU-friendly algorithm kernels. The key contributions of our work are as follows:

- We propose a linearized augmented Lagrangian formulation that incorporates augmented Lagrangian relaxation with a linearized proximal gradient descent method, which can explore the vertical and horizontal movement of cells more effectively.
- We introduce a block gradient descent method that can further enhance the parallelism of cell updates without compromising the convergence speed.
- To further enhance efficiency, we present a GPU-friendly triplefold partitioning parallelization strategy that enables our algorithm to fully utilize the parallel computing power of GPUs with minor quality overhead.

Experimental results on both ICCAD-2017 benchmarks [12] and modified ISPD-2015 benchmarks [11] demonstrate that our proposed algorithm adeptly addresses the mixed-cell-height legalization problem with high quality and stability. We achieve a $6-36\%$ improvement in overall quality score when compared to state-of-the-art legalizers on ICCAD-2017 benchmarks. On larger-scale designs with millions of cells, our proposed method shows even more significant acceleration effects, achieving a $2.25-5.99\times$ speedup compared to other state-of-the-art methods and can complete legalization within six seconds on three of the cases. Further ablation studies indicate that our proposed triplefold partitioning technique incurs less than 0.5% quality score degradation with up to $94.2\times$ speedup.

## 2 Preliminaries

### 2.1 Placement Constraints

In physical design, the purpose of the legalization phase is to adjust the positions of cells in the global placement results to ensure they meet design rule constraints. These design rule constraints can be broadly divided into hard constraints and soft constraints [12]. In this work, we focus primarily on the multi-cell-height legalization problem and the design rule constraints proposed in [12]. In addition to the two hard constraints of site alignment constraints and overlap-free constraints, the legalization results must also satisfy the following two hard constraints:

- **Power and ground (P/G) tail alignment constraints** (*hard constraint*). Cells with even cell heights must be placed in alternate rows with aligned P/G rails [11]. Cells of odd cell heights have no restriction on the row assignments because they can be flipped to align with the P/G rails [8].
- **Fence region constraints** (*hard constraint*). Cells assigned to a fence region must be placed inside the fence boundary; cells not assigned to a fence region can be placed outside the fence boundary [5].

Furthermore, we aim to minimize the number of violations of the following two routability constraints [24]:

- **Edge spacing constraints** (*soft constraint*). Due to limitations in technology and performance requirements of the circuit, certain cells must maintain a specific distance from each other. Specifically, we designate the types of the left and right sides of cells and provide the minimum required spacing between different types of cell edge types (see Fig. 1(a)).
- **Pin short / access constraints** (*soft constraint*). The signal pins of cells should not be shorted or blocked by P/G grids or IO pins [12]. Specifically, a signal pin on metal layer $k$ is considered *short* if it coincides with a P/G rail or an IO pin on the same metal

layer $k$; it is deemed *inaccessible* if it overlaps with a P/G rail or an IO pin on the adjacent metal layer $k+1$ (see Fig. 1(b)).

## 2.2 Problem Formulation

Assume the set of cells is denoted by $N$. The coordinates are scaled proportionally so that the site width is 1 and the row height is denoted as $H$. The width and height of cell $i$ are represented as $w_i$ and $h_i$ respectively, where $i \in N$. The current lower-left coordinates and the initial lower-left coordinates are denoted as $(x_i, y_i)$ and $(x_i', y_i')$, respectively. We can derive the formula for calculating displacement as follows:

$$\delta_i = |x_i - x_i'| + |y_i - y_i'|. \tag{1}$$

The weighted average displacement metric $S_{am}$ and maximum displacement metric $M_{max}$ in [12] are quantified in terms of the number of single row heights. $S_{am}$ can weight cells of different heights when measuring displacement, thereby considering the impact of cells with varying heights. We denote $M_{max} = \max_i\{\frac{\delta_i}{H}\}$ as the maximum displacement metric. Let $K \cdot H$ denote the maximum cell height ($K \in \mathbb{N}$), and $C_{kH}$ represent the set of all cells with height $kH$. The weighted average displacement metric $S_{am}$ is defined as follows:

$$S_{am} = \frac{1}{K}\sum_{k=1}^{K}\frac{1}{|C_{kH}|}\sum_{i\in C_{kH}}\frac{\delta_i}{H}. \tag{2}$$

The legalization problem can be formulated as follows:

**Problem 1** (*Mixed-Cell-Height Legalization Problem*)**.** Given a global placement result, find a legalized outcome that adheres to the following hard constraints: 1) site alignment constraints, 2) overlap-free constraints, 3) P/G tail alignment constraints, and 4) fence region constraints, while minimizing the weighted average displacement metric $S_{am}$, the maximum displacement metric $M_{max}$, and the incidence of detailed routing constraints violations, including edge spacing constraints (denoted as $N_e$) and pin access / short constraints (denoted as $N_p$).

# 3 Algorithms

In this section, we further detail our proposed LEGALM algorithm.

## 3.1 Overview

Fig. 2 depicts the workflow of our proposed LEGALM algorithm, which comprises three primary stages: 1) In the initial placement stage, we disregard the overlap-free constraint, moving the cells into or out of the fence area with minimal displacement. We refine the positions of the cells to ensure they do not intersect with placement blockages and are aligned with P/G terminals. Due to the omission of the overlap constraint, this stage is highly efficient. 2) Subsequently, as shown in Stage Two of Fig. 2, an Augmented Lagrangian Method (ALM)-based legalization algorithm is employed to produce a legal and overlap-free solution. The process illustrated in the figure is consistent with Algo. 1, which we will discuss in detail in Sec. 3.2. The backbone of this stage is a Block Gradient Descent (BGD) engine, which we will elaborate on in Sec. 3.4. 3) Finally, in Stage Three for legalization refinement, we further optimize the displacement without introducing any overflow. This is equivalent to setting extreme parameters to invoke the BGD engine, which we will explain in detail in Sec. 3.5.

## 3.2 Augmented Lagrangian Method for Mixed-cell-size Legalization

In this section, we first provide the theoretical derivation of the ALM method (Sec. 3.2.1 and Sec. 3.2.2). Then, we detail the iterative algorithm framework in Stage two of Fig. 2 (Sec. 3.2.3). Finally, we introduce more detailed parameter settings (Sec. 3.2.4).

**Algo. 1** Augmented Lagrangian Method for Mixed-Cell-Height Legalization.

**Input:** initial solution $\boldsymbol{x}^{(0)}$, maximum iteration $T$.
**Output:** legal solution $\boldsymbol{x}^{out}$.
Initialize $\sigma$ ▷ Eq. (22)
Initialize $\boldsymbol{\lambda}^{(0)}$ and $h_f$ ▷ Eq. (24) and Eq. (25)
**for** $k = 0, 1, \ldots, T-1$ **do**
  **if** $k\%K^{part} = 0$ **then**
    Update the partition scheme $R_{(\cdot)}$ and $P_{(\cdot)}$ ▷ Sec. 3.4.1
  **end if**
  **if** $k < K^{ely}$ or $k = K^{thre}$ **then**
    $\boldsymbol{x}^{(k+1)} \leftarrow \texttt{BGD}(\boldsymbol{x}^{(k)}, \boldsymbol{\lambda} = \boldsymbol{0})$ ▷ Algo. 3
  **else**
    $\boldsymbol{x}^{(k+1)} \leftarrow \texttt{BGD}(\boldsymbol{x}^{(k)}, \boldsymbol{\lambda} = \boldsymbol{\lambda}^{(k)})$ ▷ Algo. 3
  **end if**
  $of \leftarrow \texttt{CalcOverflow}(\boldsymbol{x}^{(k+1)})$
  **if** $of = 0$ **then**
    $\boldsymbol{x}^{out} \leftarrow \boldsymbol{x}^{(k+1)}$
    **break**
  **end if**
  **if** $k > K^{thre}$ and $k\%K^{h} = 0$ **then**
    Update $h_f$ ▷ Eq. (26)
  **end if**
  Update $\boldsymbol{\lambda}^{(k+1)}$ from $\boldsymbol{\lambda}^{(k)}$ ▷ Eq. (20)
**end for**

*3.2.1 Displacement Cost Function.* We employ a displacement cost function to measure the displacement cost of a cell moving to different sites. Let $S$ denote the site set, $w_{i,j}$ represent the displacement cost when the bottom-left corner of cell $i$ is located at site $j$, and $x_{i,j}$ indicates whether the bottom-left corner of cell $i$ is at site $j$ [1]. The displacement cost consists of two terms:

$$w_{i,j} = w_{i,j}^{am} + \alpha^{max} \cdot w_{i,j}^{max}, \tag{3}$$

where $w_{i,j}^{am} = \frac{|N|}{|C_{h_i}|}\delta_{i,j}$ is the *weighted average displacement cost* of cell $i$ when its bottom-left corner is at site $j$ [2], and $w_{i,j}^{max} = \max\{\delta_{i,j} - \delta^{thre}, 0\}$ is the *maximum displacement cost* of cell $i$ when its bottom-left corner is at site $j$ [3].

*3.2.2 The Augmented Lagrangian Formulation.* We provide an analytical view to formulate the legalization problem. For each cell $i$, we divide it into a set of sub-cells with a width of one and a height of row height $H$, denoted as set $\mathcal{T}_i$ ($|\mathcal{T}_i| = h_i * w_i/H$).

Let $x_{i,t,j}$ represent whether the sub-cell $t$ ($t \in \mathcal{T}_i$) of cell $i$ is located on site $j$ ($j \in S$). We can obtain the displacement cost $w_{i,t,j}$ for sub-cell $t$ of cell $i$ to be on site $j$, which is the result of the correspondingly displacement cost $w_{i,j}$ uniformly *distributed* across the sub-cells [4].

[1] $x_{i,j} \in \{0,1\}, \sum_{j\in S} x_{i,j} = 1, \forall i \in N$.
[2] $\delta_{i,j}$ represents the displacement of cell $i$ when its bottom-left corner is at site $j$. The significance of this formula is to demonstrate that $K \cdot H \cdot |N| \cdot S_{am} = \sum_{i\in N}\sum_{j\in S} w_{i,j}^{am} x_{i,j}$, which means the total weighted average displacement cost of all cells is a constant multiple of the weighted average displacement metric $S_{am}$ (see Eq. (2)).
[3] We empirically set $\alpha^{max} = 1.5$, $\delta^{thre} = 3H$.
[4] Suppose when $x_{i,t,j}$ is 1, the position of the lower-left corner of the cell $i$ is found to be at site $j'$. We have $w_{i,t,j} = \frac{w_{i,j'}}{|\mathcal{T}_i|}$. Thus, we have $\sum_{i\in N}\sum_{j\in S}\sum_{t\in\mathcal{T}_i} w_{i,t,j}x_{i,t,j} = \sum_{i\in N}\sum_{j\in S} w_{i,j}x_{i,j} = K \cdot H \cdot |N| \cdot S_{am}$.

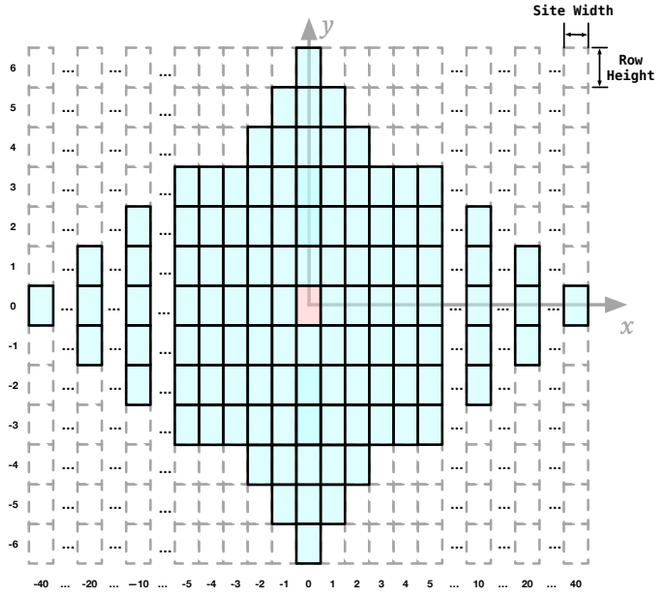



**Figure 3: The candidate positions when updating cell $i$ include a total of $D = 245$ red and blue points, with their horizontal and vertical coordinate index offsets denoted as $(\Delta_j^x, \Delta_j^y)(0 \le j < D)$. The lower left corner of cell $i$ is positioned at the red grid point. Candidate positions include the red point and all blue points. The algorithm moves the lower left corner of cell $i$ to each candidate position and computes the sum of the costs $cost_{i,t,j}^{tech}$ (Eq. (38)) on the covered sites. It selects the position with the minimum total cost among all candidate positions as the new position for cell $i$.**

Subject to the overlap-free constraints and fence region constraints, we aim to minimize the total displacement cost as follows:

$$\min \sum_{j\in S}\sum_{i\in N}\sum_{t\in\mathcal{T}_i} w_{i,t,j}x_{i,t,j}, \tag{4}$$

$$s.t.\ x_{i,t,j} \in \{0,1\}, \tag{5}$$

$$g_j(\boldsymbol{x}) = \sum_{i\in N}\sum_{t\in\mathcal{T}_i} x_{i,t,j} - 1 \le 0, \quad \forall j \in S, \tag{6}$$

$$\textit{connected sub-cell constraints}, \tag{7}$$

$$\textit{fence region constraints}, \tag{8}$$

where $g_j(\boldsymbol{x})$ represents the overflow of site $j$ under current solution $\boldsymbol{x}$. The connected sub-cell constraints require that for every cell $i$, the relative solution variables $x_{i,\cdot,\cdot}$ can uniquely restore the original shape of the cell.

To address the overlap-free constraints (6), we introduce a set of slack variables $r_j$ for each site $j \in S$:

$$\min \sum_{j\in S}\sum_{i\in N}\sum_{t\in\mathcal{T}_i} w_{i,t,j}x_{i,t,j}, \tag{9}$$

$$s.t.\ x_{i,t,j} \in \{0,1\}, \tag{10}$$

$$g_j(\boldsymbol{x}) + r_j = 0, \quad \forall j \in S, \tag{11}$$

$$r_j \ge 0, \quad \forall j \in S, \tag{12}$$

$$\textit{connected sub-cell constraints}, \tag{13}$$

$$\textit{fence region constraints}. \tag{14}$$

**Algo. 2** GD: Gradient Descent for a single cell $i$.

**Input:** cell index $i$, cell position $(x_i, y_i)$, assigned region $R$, ALM multiplier $\boldsymbol{\lambda}$.
**Output:** new cell position $(x_i^{new}, y_i^{new})$.
Unplace cell $i$ and update $g(\cdot)$ ▷ Eq. (6)
**parallel for** $j = 0, 1, \cdots, D-1$ **do**
  $x'_j, y'_j \leftarrow x_i + \Delta_j^x, y_i + \epsilon_i \cdot H \cdot \Delta_j^y$ ▷ Fig. 3
  $C_j \leftarrow \textsc{CalcCost}(x'_j, y'_j, R, \boldsymbol{\lambda})$ ▷ Eq. (38)
**end for**
$j' = \text{argmin}_j\, C_j$ ▷ Parallel reduction
$x_i^{new}, y_i^{new} \leftarrow x'_{j'}, y'_{j'}$
Place cell $i$ and update $g(\cdot)$ ▷ Eq. (6)
Return $(x_i^{new}, y_i^{new})$

We then relax the overlap-free constraints (6) and obtain the augmented Lagrangian function [25]:

$$\begin{aligned}\mathcal{L}(\boldsymbol{x},\boldsymbol{r},\boldsymbol{\lambda}) = &\sum_{j\in S}\sum_{i\in N}\sum_{t\in\mathcal{T}_i} w_{i,t,j}x_{i,t,j} \\ &+ \sum_{j\in S}\lambda_j\left[(g_j(\boldsymbol{x}) + r_j) + \frac{\sigma}{2}(g_j(\boldsymbol{x}) + r_j)^2\right] \\ &+ I_{\mathcal{X}}(\boldsymbol{x}),\end{aligned} \tag{15}$$

where $\boldsymbol{\lambda}$ is the Lagrange multiplier, and $\sigma$ is the penalty factor. $I_{\mathcal{X}}(\boldsymbol{x})$ is the indicator function for the feasible solution space $\mathcal{X}$, which equals 0 when $\boldsymbol{x}$ satisfies the connected sub-cell constraints and the fence region constraints; otherwise it equals $+\infty$.

The augmented Lagrangian method gradually approaches the optimal solution through iterative solving. In the $k$-th iteration of the augmented Lagrangian method, given the multiplier $\boldsymbol{\lambda}^{(k)}$, we need to solve the following optimization problem:

$$\min_{\boldsymbol{x},\boldsymbol{r}} \mathcal{L}(\boldsymbol{x},\boldsymbol{r},\boldsymbol{\lambda}^{(k)}). \tag{16}$$

It is observed that, given $\boldsymbol{x}$, the optimal value of $\boldsymbol{r}$ can be solved through the *elimination method* [5] [3]:

$$r_j = \max(0, -\frac{1}{\sigma} - g_j(\boldsymbol{x})). \tag{17}$$

In other words, the optimal value of $\boldsymbol{r}$ can be expressed in terms of $\boldsymbol{x}$, so we can omit $\boldsymbol{r}$ from the optimization variables in the following formula and define $\psi(\boldsymbol{x},\boldsymbol{\lambda})$ as:

$$\psi(\boldsymbol{x},\boldsymbol{\lambda}) = \sum_{j\in S}\sum_{i\in N}\sum_{t\in\mathcal{T}_i} w_{i,t,j}x_{i,t,j} + \sum_{j\in S}\lambda_j\left[(g_j(\boldsymbol{x}) + r_j) + \frac{\sigma}{2}(g_j(\boldsymbol{x}) + r_j)^2\right]. \tag{18}$$

Within each augmented Lagrangian iteration, we need to solve the following optimization problem:

$$\boldsymbol{x}^{(k+1)} = \text{argmin}_{\boldsymbol{x}}\, \mathcal{L}(\boldsymbol{x},\boldsymbol{\lambda}^{(k)}) = \text{argmin}_{\boldsymbol{x}}\, \psi(\boldsymbol{x},\boldsymbol{\lambda}^{(k)}) + I_{\mathcal{X}}(\boldsymbol{x}). \tag{19}$$

After the iteration, we update the multiplier $\boldsymbol{\lambda}$ by using the KKT conditions [3]:

$$\lambda_j^{(k+1)} = \max\left(\lambda_j^{(k)} + h_f \cdot \left[(g_j(\boldsymbol{x}) + r_j) + \frac{\sigma}{2}(g_j(\boldsymbol{x}) + r_j)^2\right], 0\right), \tag{20}$$

where $h_f$ is the weighting factor.

*3.2.3 Summary of ALM-based Legalization.* Algo. 1 [6] summarizes the iterative algorithm framework derived from the above theoretical derivation, which is consistent with Stage Two in Fig. 2. We will detail the parameter settings later in Sec. 3.2.4. The underlying philosophy of this framework is similar to that of the analytical global placement in [9, 19, 20], where the primary focus during the early iterations is to

[5] Due to space limitations, we omit the derivation process here.
[6] We empirically set $K^{part} = 50$, $K^{ely} = 2$, $K^{thre} = 300$, and $K^h = 100$.

optimize the displacement cost. As the iterations progress, we gradually enforce the overlap-free constraints until an overlap-free solution is achieved.

Firstly, we initialize the parameters of the optimization problem (line 3-4). Then, we proceed to the iterative solution process. In each iteration, we first determine whether there is a need to update the partition scheme (which will be detailed in Sec. 3.4.1). Subsequently, a Block Gradient Descent (BGD) engine (Algo. 3) is employed to solve the sub-problem Eq. (19).

It should be noted that during the first $K^{ely}$ iterations (line 9-10), we set $\boldsymbol{\lambda}$ to zero, which is equivalent to disregarding the overflow and focusing solely on optimizing the displacement cost. This approach is beneficial for fully optimizing the displacement in the early stages of the iterative process. Additionally, a similar operation is performed at the $K^{thre}$-th iteration, which is akin to restarting the optimizer during the optimization process, aiding in escaping from local optima. If an overlap-free solution is obtained, we terminate the iterations and return a legal solution (line 14-18). After the $K^{thre}$-th iteration, we begin to incrementally update the weighting factor $h_f$ to gradually strengthen the overlap-free constraints (line 20). Meanwhile, we update $\boldsymbol{\lambda}$ after each iteration (line 22).

*3.2.4 Parameter Initialization and Updating.* The intuition for initializing $\sigma$ is to evaluate the current congestion level. The higher the congestion, the larger $\sigma$ needs to be increased to eliminate congestion. Let $d_i(\boldsymbol{x})$ represent the sum of demands of all sites covered by cell $i$,

$$d_i(\boldsymbol{x}) = \sum_{t\in\mathcal{T}_i}\sum_{j\in S}(1+g_j(\boldsymbol{x}))x_{i,t,j}. \tag{21}$$

The larger the average $d_i(\boldsymbol{x})$ of cells, the higher the current congestion level. We use this value to initialize $\sigma$[7]:

$$\sigma = \max\{1, \alpha^{\sigma}\cdot\frac{\sum_{i\in N} d_i(\boldsymbol{x}^{(0)})}{|N|}\}. \tag{22}$$

The approach for initializing $\boldsymbol{\lambda}$ is to balance the ratio between the displacement cost and the augmented Lagrangian function term. Based on this, following the definition of the augmented Lagrangian function term, we define $ad_i(\boldsymbol{x})$ to represent the augmented demand term of cell $i$:

$$ad_i(\boldsymbol{x}) = \sum_{t\in\mathcal{T}_i}\sum_{j\in S}\left[1+(1+g_j(\boldsymbol{x}))+\frac{\sigma}{2}(1+g_j(\boldsymbol{x}))^2\right]x_{i,t,j}. \tag{23}$$

We assume that the final cell displacement distance is on the order of row height $H$, and initialize $\boldsymbol{\lambda}$ through the average ratio of the expected displacement to the augmented demand term. The initialization of $\boldsymbol{\lambda}$ is as follows [8]:

$$\lambda_j^{(0)} \leftarrow \widetilde{\lambda} = \alpha^{\widetilde{\lambda}}\cdot\frac{\sum_{i\in N} H/ad_i(\boldsymbol{x}^{(0)})}{|N|}. \tag{24}$$

In the later stages of iteration, we also need to gradually increase the weighting factor $h_f$ to progressively strengthen the penalty for violating the overflow constraint and gradually eliminate overflow (line 4 and 20) as follows [9]:

$$h_f^{(0)} \leftarrow \widetilde{\lambda}, \tag{25}$$

$$h_f^{(k+1)} \leftarrow h_f^{(k)} + \alpha^{h_f}\cdot\widetilde{\lambda}. \tag{26}$$

In the following two sections, Sec. 3.3 first introduces how to derive the update of a single cell from the perspective of linearized proximal gradient method and summarizes it into the function GD (see Algo. 2). Then, Sec. 3.4 explains how to schedule multiple cells' GD function using the block gradient descent method and summarizes it into the engine BGD (see Algo. 3).

[7] $\alpha^{\sigma}$ is set to 3.

[8] $\alpha^{\widetilde{\lambda}}$ is set to 1.5.

[9] We empirically set $\alpha^{h_f} = 0.2$.

**Algo. 3** BGD: Block Gradient Descent for a series of cell partition set $\{P_l\}_{l=0}^{L-1}$ and the corresponding sub-partition set $\{R_l\}_{l=0}^{L-1}$.

1: **Input:** current solution $\boldsymbol{x}$, ALM multiplier $\boldsymbol{\lambda}$, cell partition set $\{P_l\}_{l=0}^{L-1}$, sub-partition set $\{R_l\}_{l=0}^{L-1}$.
2: **Output:** new solution $\boldsymbol{x}^{new}$.
3: **parallel for** $l = 0, 1, \cdots, L-1$ **do**
4: Sort cells in $P_l$ by $d_i(\boldsymbol{x})$ in descending order. ▷ Eq. (21)
5: **for** $i \in P_l$ **do**
6: $(x_i, y_i) \leftarrow$ *retrieve the lower left corner of cell i from* $\boldsymbol{x}$
7: $(x_i^{new}, y_i^{new}) \leftarrow \mathrm{GD}(i, x_i, y_i, R_l, \boldsymbol{\lambda})$ ▷ Algo. 2
8: *Update* $x_{i,\cdot,\cdot}^{new}$ *from* $(x_i^{new}, y_i^{new})$
9: **end for**
10: **end for**
11: return $\boldsymbol{x}^{new}$

## 3.3 Linearized Proximal Gradient Method

In this section, we detail how to solve the sub-problem Eq. (19). Firstly, Sec. 3.3.1 presents the theoretical derivation of the linearized proximal gradient method, and then Sec. 3.3.2 introduces the routability constraints penalty. Finally, Sec. 3.3.3 summarizes and provides the pseudocode for the algorithm implementation.

*3.3.1 Theoretical Derivation.* Given $\boldsymbol{\lambda}$, denote $f(\boldsymbol{x})$ as $\psi(\boldsymbol{x}, \boldsymbol{\lambda})$ in Eq. (18). We aim at solving the following optimization problem:

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}) + I_{\mathcal{X}}(\boldsymbol{x}). \tag{27}$$

Note that $f(\cdot)$ is differentiable and the proximal mapping operator of $\mathcal{X}$ is easy to obtain, we adopt the *proximal linear method* [2] as a linearization technique to resolve Eq. (27). The proximal mapping operator of $\mathcal{X}$ is defined as follows:

$$P_{\mathcal{X}}(\boldsymbol{v}) = \arg\min_{\boldsymbol{u}\in\mathcal{X}} \|\boldsymbol{u}-\boldsymbol{v}\|_2^2, \tag{28}$$

where $P_{\mathcal{X}}(\boldsymbol{v})$ is a set. We then solve for $\boldsymbol{x}$ through an iterative approach [2]. The update formula for the $k$-th iteration is:

$$\boldsymbol{x}^{(k+1)} \in P_{\mathcal{X}}\left(\boldsymbol{x}^{(k)} - \tau\nabla f(\boldsymbol{x}^{(k)})\right), \tag{29}$$

where $\tau$ is the step size. To solve Eq. (28) and Eq. (29), we first simplify the proximal mapping operator Eq. (28) and obtain

$$\begin{aligned} P_{\chi}(\boldsymbol{v}) &= \arg\min_{\boldsymbol{u}\in\chi} \|\boldsymbol{u}-\boldsymbol{v}\|_2^2 && (30)\\ &= \arg\min_{\boldsymbol{u}\in\chi} \|\boldsymbol{u}\|_2^2 - 2\boldsymbol{v}^T\boldsymbol{u} + \|\boldsymbol{v}\|_2^2 && (31)\\ &= \arg\min_{\boldsymbol{u}\in\chi} \mathbf{1}^T\boldsymbol{u} - 2\boldsymbol{v}^T\boldsymbol{u}, \text{ as } u_i \in \{0,1\} \text{ when } \boldsymbol{u}\in\chi && (32)\\ &= \operatorname*{argmin}_{\boldsymbol{u}\in\chi}(\mathbf{1}-2\boldsymbol{v})^T\boldsymbol{u} && (33)\\ &= \operatorname*{argmin}_{\boldsymbol{u}\in\chi}\boldsymbol{c}^T\boldsymbol{u} \quad (\text{denote } \mathbf{1}-2\boldsymbol{v} \text{ as } \boldsymbol{c}) && (34) \end{aligned}$$

We can observe that due to the characteristics of the solution space $\mathcal{X}$, the proximal mapping operator essentially performs a linearization operation. In other words, given $\mathbf{v}$, $P_{\chi}(\boldsymbol{v})$ finds the $\boldsymbol{u}$ that minimizes the inner product with $\boldsymbol{c} = \mathbf{1} - 2\boldsymbol{v}$.

Then we simplify Eq. (29). The derivative of $f(\cdot)$ is[10]:

$$\nabla_{x_{i,t,j}} f(\boldsymbol{x}) = w_{i,t,j} + \lambda_j T_j, \tag{35}$$

[10] We omit the derivation process here.

where $T_j = \max\{1 + \sigma g_j(\boldsymbol{x}), 0\}$. Substituting Eq. (35) into Eq. (29) and Eq. (34), we can obtain

$$c_{i,t,j} = 1 - 2x_{i,t,j} + 2\tau(w_{i,t,j} + \lambda_j T_j). \tag{36}$$

According to the definition of the proximal gradient method, we are looking for a new feasible solution $\boldsymbol{x}^{(k+1)}$ near the current solution $\boldsymbol{x}^{(k)}$ [11], such that the inner product $\boldsymbol{c}^T\boldsymbol{x}^{(k+1)}$ is minimized. In practice, we set $\tau$ sufficiently large, so we can disregard the term $1 - 2x_{i,t,j}$ in Eq. (36) and focus on the relative magnitudes of $w_{i,t,j} + \lambda_j T_j$ when comparing different candidate positions. Therefore, we can define the cost function as

$$cost_{i,t,j} = w_{i,t,j} + \lambda_j T_j. \tag{37}$$

*3.3.2 Routability Constraints Penalty.* To further consider routability constraints, we incorporate a penalty term for violations of routability constraints into the cost function Eq. (37). Let $R_{i,t,j}$ denote the total number of violations of edge spacing constraints, pin short, and pin access constraints when subcell $t$ of cell $i$ (with $t \in \mathcal{T}_i$) is located at site $j$, while keeping the solutions of other cells unchanged. The cost function considering routability constraints is as follows [12]:

$$cost^{tech}_{i,t,j} = cost_{i,t,j} + p^{tech} \cdot H \cdot R_{i,t,j}. \tag{38}$$

*At this point, $cost^{tech}_{i,t,j}$ precisely represents the cost given by the linearized proximal gradient method when subcell $t$ of cell $i$ is located at site $j$.*

*3.3.3 Summary of Linearized Proximal Gradient Method.* Algo. 2 summarizes gradient descent step for each cell[13]. We first unplace cell $i$, and decrease the overflow function of the covered sites by 1 (line 3). Then we enumerate the nearby candidate positions (line 4-7) parallelly, where $\epsilon_i$ is 2 if and only if the height of cell $i$ is an even multiple of $H$; otherwise it is 1[14]. For each enumerated position $(x'_j, y'_j)$, if it is not legal, i.e., not within the solution space $\mathcal{X}$ or not entirely within the region $R$, the value of $C_j$ is $+\infty$; otherwise calculate the sum of the costs of the covered sites according to Eq. (38) as $C_j$. Soft constraints violations are only considered within the assigned region $R$. Subsequently, we obtain the candidate position with the minimum moving cost (line 8) and the new position of cell $i$ (line 9). Finally, we place cell $i$ and increase the overflow function of the newly covered sites by 1 (line 10). Note that our enumeration includes the initial position, which is guaranteed to be legal (see Fig. 3), so it is certain that we can move to a legal solution.

## 3.4 Block Gradient Descent Method

Algo. 2 provides the updating method for each cell, and we need to schedule the updates of all cells while ensuring the algorithm converges within a reasonable time. 1) One extreme is to update each cell by Algo. 2 sequentially, which is actually equivalent to the block gradient descent method with the smallest granularity block, i.e., each block consists of only one cell. The advantage of this method is that fewer ALM iterations are required, but within each iteration, the updates of cells are not independent of each other, which is not conducive to parallel computing acceleration. 2) On the other extreme, updating all cells based on the result of the last ALM iteration $\boldsymbol{x}^{(k)}$ simultaneously has the advantage that the updates of cells within the ALM iteration are independent and thus can be processed in parallel, but the disadvantage is that the number of ALM iterations is higher.

[11] We empirically define the surrounding position for each cell as shown in Fig. 3.
[12] We empirically set $p^{tech} = 1$. Incorporating highly discrete routability constraints penalty can disrupt the algorithm's global view, but due to their relatively small quantity, they actually have little impact on the algorithm's performance.
[13] We will detail the assigned region $R$ in Sec. 3.4.
[14] The purpose of this is to ensure that cell $i$ still satisfies the P/G alignment constraint before and after moving.

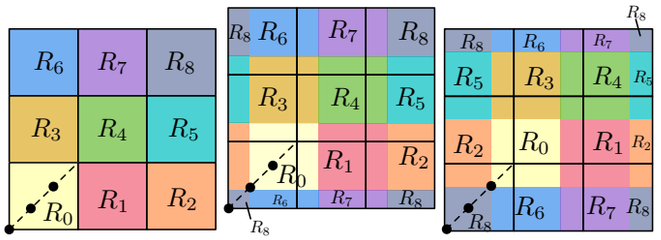



**Figure 4: This diagram shows a simplified illustration of triplefold partitioning. As depicted in Fig. 4(a), the entire layout is divided into $L = 3 \times 3$ grids (solid black lines). Fig. 4(a)-Fig. 4(c) represent triplefold partitioning schemes, where each subfigure contains $L$ colors representing $L$ sub-regions $\{R_l\}_{l=0}^{L-1}$ (note that sub-regions may not be connected). The yellow sub-region has its bottom-left corner at the trisection point of the diagonal of the bottom-left grid in Fig. 4(b)-Fig. 4(c).**

*3.4.1 Triplefold Partitioning (TP) Algorithm.* To fully combine the advantages of these two methods, we propose a triplefold partitioning algorithm method that allows for parallel processing while minimizing the mutual influence between position updates within blocks as illustrated in Fig. 4[15] and Algo. 3.

Assume the layout size is $S_w \times S_h$, we divide the entire layout into $\lfloor S_w/x^{hint} \rfloor \times \lfloor S_h/y^{hint} \rfloor$ grids of approximately equal size [16]. Based on this grid division, we then construct three schemes for creating sub-regions $\{R_l\}_{l=0}^{L-1}$. Take Fig. 4(a) as an example, we divide the entire layout into $L$ non-overlapping sub-regions $\{R_l\}_{l=0}^{L-1}$ and construct corresponding cell subsets $\{P_l\}_{l=0}^{L-1}$. Cell $i$ belongs to $P_l$ if and only if the current position of cell $i$ is entirely within sub-region $R_l$. Cells that are in multiple sub-regions are not considered in the current partition scheme. Sub-regions and their corresponding cell subsets are processed in parallel (line 3–10). Within each sub-region $R_l$, the cells in $P_l$ are sorted by their demand $d_i(\boldsymbol{x})$ (Eq. (21)) in descending order (line 4), and updated sequentially (line 5–9).

During each iteration of our augmented Lagrangian algorithm (line 7 in Algo. 1), $R_{(\cdot)}$ and its corresponding $P_{(\cdot)}$ rotate among these three partition schemes to ensure that all cells have the opportunity to be updated. It can be demonstrated that with at least three partition schemes, a cell will be entirely contained within a certain sub-region in at least one of the partition schemes, allowing it to move freely as the partition schemes rotate.

*3.4.2 GPU-Accelerated Kernels.* To fully harness the power of massive parallelization on GPUs, we implement the core algorithm `BGD` as a GPU-optimized engine. We configure the GPU kernel with $L$ blocks, each containing $D$ threads. This structure aligns perfectly with our algorithmic design.

Each GPU block $i$ is assigned the task of updating the sub-partition set $R_i$ and its corresponding cell partition $P_i$ (as shown in line 7 of Algo. 3). This block-level parallelism allows for simultaneous processing of different sub-regions of the layout.

Within each block, we leverage thread-level parallelism. Each thread $j$ is responsible for calculating the cost at a distinct candidate position (line 4-7 in Algo. 2). The computed cost values (line 6 in Algo. 2) are stored in a shared memory array, facilitating fast inter-thread communication. To determine the optimal position, we employ a parallel reduction algorithm

[15] The $3 \times 3$ grid division shown here is for simplification and illustrative purposes only. In practice, the grid division will adapt to the layout's shape.
[16] $(x^{hint}, y^{hint})$ serve as hints for the grid size, set to $(250, 25H)$ in our implementation.

(line 8 in Algo. 2) to efficiently find the minimum cost value across all threads.

This two-tiered parallelization strategy—at both the block and thread levels—enables us to efficiently process large layouts and significantly accelerate the legalization process.

### 3.5 LG Refinement

Following the ALM-based legalization process, we obtain an overlap-free solution. To further enhance this result, we employ a refinement phase utilizing the same BGD engine, but with a crucial modification: we set $\boldsymbol{\lambda} = +\infty$. This adjustment, effectively invoking $\mathtt{BGD}(\boldsymbol{x}, \boldsymbol{\lambda} = +\infty)$, is tantamount to imposing an infinite penalty on overflow when evaluating candidate positions. Consequently, this approach rigorously preserves the overlap-free property of the solution while allowing for positional optimizations.

The refinement procedure iteratively applies the three partitioning schemes. Through empirical analysis, we observed that the marginal improvements in solution quality begin to diminish after approximately 10 iterations for each partition scheme. Therefore, we adopt this number of iterations as the optimal trade-off between refinement quality and computational efficiency.

## 4 Experimental Results

We implement LEGALM with C++/CUDA for GPU and C++/OpenMP for multi-threaded CPU, respectively. We adopt PyTorch for agile GPU memory management and NVIDIA CUB for parallel sorting and reduction on GPU. We conduct experiments on a CentOS 7.9 server with two Intel Xeon Platinum 8358 CPUs (2.60GHz, 28 cores) and one NVIDIA A800 GPU. The benchmarks are from the ICCAD-2017 mixed-cell-height legalization contest [12] with routability and fence region constraints, and the modified ISPD-2015 benchmarks where 10% of the single row cells are modified to double-row cells as stated in [11].

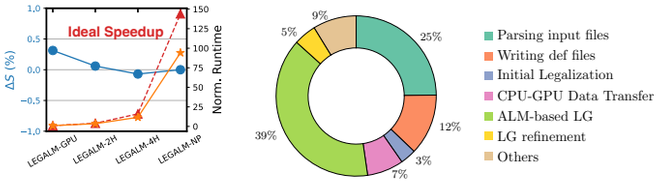
*4.0.1 Comparison with State-of-the-Art Placers on ICCAD-2017 Benchmarks.* Table 1[17] summarizes the statistics of ICCAD-2017 benchmarks as well as the comparison with the state-of-the-art placers. We compare our proposed algorithms on both CPU (LEGALM-CPU) and GPU (LEGALM-GPU) with state-of-the-art methods [6, 17, 26]. We do not compare with [23] because it does not consider the routability constraints. To fairly compare the overall performance of different algorithms, we adopt the quality score $S$ from [17] as follows:

$$S = \left(1 + S_{hpwl} + \frac{N_p + N_e}{|N|}\right)\left(1 + \frac{M_{max}}{100}\right) S_{am} \tag{39}$$

where $S_{hpwl}$ is the increase ratio of HPWL, and the definition of other symbols are stated in Sec. 2.2. Note that all runtime data except those marked with † are the results run on our server. [6] is executed on our server with four threads as the same setting in its open-source release. We can see that LEGALM-GPU achieves the best quality score among all compared algorithms, i.e., 36% better than [17], 25% better than [26], and 6% better than [6] with 3.83× speed-up on average.

To compare the performance with [6] under the same number of CPU cores, we implement a CPU parallel version LEGALM-CPU with four threads. In this version, we change the way the grid is divided in Sec. 3.4.1 to divide it into $2 \times 2$ grids, so that each CPU thread is responsible for one sub-region. The experimental results show that under the same number of CPU threads, LEGALM-CPU and [6] have similar runtime, and the quality score is 36%, 25%, and 6% better than [17], [26], and [6], respectively. Meanwhile, compared to LEGALM-CPU, LEGALM-GPU achieves a 3.98× speedup with minor quality degradation. We attribute these benefits to the linearized proximal gradient method, which provides a global view that can effectively reduce overflow with minor displacement overhead.

[17]The runtime data in this table exclude the file I/O time, GPU data transfer time, and GPU memory allocation time, as in practice, all the data are already in memory when running the entire backend flow.




(a) (b)

**Figure 5: (a) The relative quality score change $\Delta S$ and normalized runtime of LEGALM-GPU and the three baselines on des_perf_b_md2. (b) The runtime breakdown of LEGALM-GPU on mgc_superblue12.**

Table 2 further shows the breakdown of quality scores of LEGALM-GPU and other legalizers. We can see that LEGALM-GPU consistently achieves better average displacement $S_{am}$ than other placers on all cases with 33.2%, 25.0%, and 4.4% better than [17], [26], and [6], respectively. The HPWL variation $S_{hpwl}$ of LEGALM-GPU is 29%, 34%, and 13% better than that in [17], [26], and [6], respectively. LEGALM-GPU also achieves 10% better maximum displacement $M_{max}$ than [6, 17, 26]. Although the number of our soft constraint violations $N_p$ is higher than that of other legalizers, our other metrics are significantly lower than those of the legalizers, thereby resulting in a better overall quality. This indicates that our analytical constraint optimization problem modeling can effectively optimize the objective under overlap-free constraints.

*4.0.2 Comparison with State-of-the-Art Placers on Large-scale Benchmarks.* To demonstrate the scalability of our approach, we further conduct experiments on five superblue series large designs from modified ISPD-2015 benchmarks. Table 3 summarizes the statistics of the five benchmarks as well as the comparison with three state-of-the-art placers [6, 8, 11]. The experimental results show that for million-scale designs, LEGALM-GPU demonstrates more significant acceleration effects and better scalability compared to the CPU version LEGALM-CPU. Meanwhile, LEGALM-GPU achieves 2.25× to 5.99× speedup compared to other methods [6, 8, 11], and can complete legalization within ten seconds for three cases. This indicates that LEGALM-GPU has good scalability for large-scale designs.

*4.0.3 Ablation Study for Triplefold Partitioning Algorithm.* To demonstrate the effectiveness of our proposed triplefold partitioning algorithm, we construct three baselines for ablation study on GPU, namely LEGALM-2H, LEGALM-4H, and LEGALM-NP. LEGALM-2H indicates the use of a doubled grid size hint, i.e., $(2x^{hint}, 2y^{hint})$ (see Sec. 3.4.1) to construct the grid mesh. LEGALM-4H follows the same logic. The larger the grid size hint, the greater the range within which cells in the sub-region can move, but the lower the degree of parallelism. LEGALM-NP signifies the non-use of the triplefold partitioning algorithm.

Fig. 5(a) presents the relative quality score change $\Delta S$ and normalized runtime of LEGALM-GPU and the three baselines on des_perf_b_md2, the case with the most fence regions. We can observe that compared to LEGALM-NP, LEGALM-GPU only incurs less than 0.5% quality score degradation, yet it achieves 94.2× speedup. Additionally, LEGALM-GPU, LEGALM-2H, and LEGALM-4H all achieve an acceleration ratio close to the ideal. This indicates that the TP algorithm can significantly speed up legalization with minor quality degradation.

*4.0.4 Runtime Breakdown.* Fig. 5(b) shows the runtime breakdown of LEGALM-GPU on the largest case mgc_superblue12. We can see

**Table 1: Benchmark statistics, quality scores, and runtime comparison on ICCAD–2017 benchmarks [12].**

| Case | #Cells of Different Heights ($H$) | | | | Den. (%) | #Regions | Runtime (s) | | | | | Score $S$ | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | | | [17]† | [26]† | [6] | LEGALM-CPU | LEGALM-GPU | [17] | [26] | [6] | LEGALM-CPU | LEGALM-GPU |
| des_perf_1 | 112644 | 0 | 0 | 0 | 90.6 | 0 | 7.91 | 15.93 | 27.71 | 29.41 | 7.51 | 1.10 | 0.91 | 0.92 | 0.66 | 0.66 |
| des_perf_a_md1 | 103589 | 4699 | 0 | 0 | 55.1 | 4 | 6.70 | 4.54 | 29.23 | 23.29 | 8.38 | 1.87 | 1.6 | 1.28 | 1.18 | 1.20 |
| des_perf_a_md2 | 105030 | 1086 | 1086 | 1086 | 55.9 | 4 | 6.47 | 6.03 | 44.66 | 29.18 | 16.64 | 2.12 | 1.9 | 1.17 | 1.11 | 1.12 |
| des_perf_b_md1 | 106782 | 5862 | 0 | 0 | 55.0 | 12 | 5.95 | 3.65 | 11.91 | 15.27 | 20.34 | 0.82 | 0.78 | 0.66 | 0.64 | 0.65 |
| des_perf_b_md2 | 101908 | 6781 | 2260 | 1695 | 64.7 | 12 | 6.21 | 3.11 | 14.15 | 16.45 | 1.11 | 0.91 | 0.8 | 0.71 | 0.70 | 0.70 |
| edit_dist_1_md1 | 118005 | 7994 | 2664 | 1998 | 67.4 | 0 | 6.34 | 4.06 | 20.01 | 33.93 | 2.68 | 0.81 | 0.83 | 0.67 | 0.65 | 0.63 |
| edit_dist_a_md2 | 115066 | 7799 | 2599 | 1949 | 59.4 | 1 | 7.78 | 4.42 | 16.76 | 22.61 | 2.22 | 0.82 | 0.75 | 0.68 | 0.67 | 0.67 |
| edit_dist_a_md3 | 119616 | 2599 | 2599 | 2599 | 57.2 | 1 | 10.39 | 16.92 | 47.52 | 41.40 | 19.21 | 1.14 | 1.05 | 0.88 | 0.79 | 0.79 |
| fft_2_md2 | 28930 | 2117 | 705 | 529 | 82.7 | 0 | 1.82 | 1.28 | 5.66 | 5.56 | 1.74 | 1.11 | 1.01 | 0.82 | 0.75 | 0.68 |
| fft_a_md2 | 27431 | 2018 | 672 | 504 | 32.3 | 0 | 1.60 | 0.81 | 3.31 | 3.09 | 0.51 | 0.86 | 0.77 | 0.74 | 0.74 | 0.75 |
| fft_a_md3 | 28609 | 672 | 672 | 672 | 31.2 | 0 | 1.86 | 0.82 | 2.67 | 3.03 | 0.39 | 0.68 | 0.61 | 0.59 | 0.59 | 0.59 |
| pci_bridge32_a_md1 | 26680 | 1792 | 597 | 448 | 49.5 | 4 | 1.76 | 1.13 | 2.15 | 2.48 | 0.70 | 1.09 | 0.98 | 0.93 | 0.91 | 0.92 |
| pci_bridge32_a_md2 | 25239 | 2090 | 1194 | 994 | 57.7 | 4 | 1.50 | 3.35 | 10.81 | 5.45 | 2.12 | 1.10 | 1.11 | 0.91 | 0.85 | 0.85 |
| pci_bridge32_b_md1 | 26134 | 1756 | 585 | 439 | 26.6 | 3 | 1.72 | 1.01 | 3.47 | 5.02 | 0.88 | 1.34 | 1.28 | 1.16 | 1.13 | 1.14 |
| pci_bridge32_b_md2 | 28038 | 292 | 292 | 292 | 18.3 | 3 | 4.26 | 0.90 | 3.84 | 4.36 | 1.69 | 1.31 | 1.27 | 1.01 | 1.00 | 1.01 |
| pci_bridge32_b_md3 | 27452 | 292 | 585 | 585 | 22.2 | 3 | 2.15 | 1.41 | 5.78 | 6.44 | 1.92 | 1.61 | 1.58 | 1.09 | 1.07 | 1.09 |
| Ratio | - | - | - | - | - | - | 1.45 | 1.03 | 3.83 | 3.98 | 1.00 | 1.36 | 1.25 | 1.06 | 1.00 | 1.00 |

† The runtime data is quoted from the publications for reference.

**Table 2: Detailed breakdown of quality scores on ICCAD–2017 benchmarks [12].**

| Case | $S_{hpwl}$ (%) | | | | $M_{max}$ | | | | $S_{am}$ | | | | $N_p$ | | | | $N_e$ | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | [17] | [26] | [6] | LEGALM-GPU | [17] | [26] | [6] | LEGALM-GPU | [17] | [26] | [6] | LEGALM-GPU | [17] | [26] | [6] | LEGALM-GPU | [17] | [26] | [6] | LEGALM-GPU |
| des_perf_1 | 10.80 | 9.16 | 8.28 | 5.40 | 8.4 | 6.6 | 10.5 | 4.3 | 0.903 | 0.781 | 0.758 | 0.582 | 1815 | 1279 | 1353 | 3027 | 0 | 0 | 0 | 0 |
| des_perf_a_md1 | 3.64 | 3.18 | 2.91 | 2.59 | 60.7 | 60.7 | 60.7 | 60.7 | 1.122 | 0.966 | 0.772 | 0.725 | 90 | 52 | 174 | 668 | 0 | 0 | 0 | 0 |
| des_perf_a_md2 | 3.71 | 3.71 | 3.04 | 2.66 | 48.1 | 40.4 | 40.4 | 40.4 | 1.380 | 1.308 | 0.804 | 0.775 | 188 | 58 | 279 | 785 | 0 | 0 | 0 | 0 |
| des_perf_b_md1 | 2.51 | 2.51 | 2.28 | 2.19 | 10.0 | 9.0 | 10.6 | 11.6 | 0.725 | 0.696 | 0.578 | 0.571 | 168 | 122 | 178 | 510 | 0 | 0 | 0 | 0 |
| des_perf_b_md2 | 2.88 | 2.41 | 1.99 | 1.88 | 23.3 | 19.1 | 19.1 | 18.9 | 0.718 | 0.652 | 0.585 | 0.577 | 26 | 0 | 180 | 783 | 0 | 0 | 0 | 0 |
| edit_dist_1_md1 | 2.10 | 2.35 | 1.86 | 1.59 | 5.7 | 7.0 | 4.8 | 4.7 | 0.752 | 0.761 | 0.610 | 0.587 | 45 | 0 | 516 | 1927 | 0 | 0 | 3171 | 0 |
| edit_dist_a_md2 | 1.48 | 1.48 | 1.18 | 1.15 | 16.4 | 16.4 | 16.4 | 16.4 | 0.697 | 0.634 | 0.573 | 0.568 | 42 | 0 | 179 | 712 | 0 | 0 | 0 | 0 |
| edit_dist_a_md3 | 2.25 | 2.82 | 1.53 | 1.33 | 31.4 | 23.3 | 29.8 | 23.5 | 0.837 | 0.831 | 0.662 | 0.626 | 1342 | 130 | 1012 | 1782 | 0 | 0 | 0 | 0 |
| fft_2_md2 | 14.58 | 16.83 | 9.72 | 7.67 | 7.1 | 6.8 | 4.9 | 3.8 | 0.905 | 0.818 | 0.642 | 0.586 | 196 | 0 | 344 | 1124 | 0 | 0 | 3307 | 0 |
| fft_a_md2 | 1.50 | 1.50 | 1.50 | 1.39 | 34.3 | 34.3 | 34.3 | 34.3 | 0.631 | 0.567 | 0.544 | 0.539 | 4 | 0 | 8 | 569 | 0 | 0 | 0 | 0 |
| fft_a_md3 | 1.04 | 2.09 | 1.73 | 1.56 | 11.3 | 11.0 | 11.5 | 11.6 | 0.605 | 0.537 | 0.518 | 0.512 | 2 | 0 | 13 | 586 | 0 | 0 | 0 | 0 |
| pci_bridge32_a_md1 | 5.57 | 3.37 | 3.75 | 3.55 | 45.9 | 42.6 | 42.6 | 42.6 | 0.712 | 0.662 | 0.624 | 0.619 | 25 | 0 | 64 | 145 | 0 | 0 | 0 | 0 |
| pci_bridge32_a_md2 | 6.08 | 6.08 | 5.00 | 4.05 | 18.1 | 18.1 | 18.2 | 18.4 | 0.872 | 0.879 | 0.726 | 0.680 | 183 | 161 | 277 | 494 | 0 | 0 | 0 | 0 |
| pci_bridge32_b_md1 | 2.86 | 4.37 | 3.00 | 2.81 | 51.4 | 51.4 | 52.0 | 51.4 | 0.853 | 0.818 | 0.740 | 0.730 | 3 | 0 | 39 | 216 | 0 | 0 | 0 | 0 |
| pci_bridge32_b_md2 | 2.54 | 2.54 | 3.44 | 3.30 | 61.7 | 54.6 | 54.6 | 54.6 | 0.785 | 0.794 | 0.630 | 0.630 | 5 | 0 | 26 | 138 | 0 | 0 | 0 | 0 |
| pci_bridge32_b_md3 | 4.42 | 4.42 | 4.01 | 3.89 | 49.8 | 49.8 | 52.0 | 51.4 | 1.031 | 1.009 | 0.686 | 0.686 | 38 | 9 | 52 | 186 | 0 | 0 | 0 | 0 |
| Ratio | 1.29 | 1.34 | 1.13 | 1.00 | 1.1 | 1.1 | 1.1 | 1.0 | 1.332 | 1.250 | 1.044 | 1.000 | - | - | - | - | - | - | - | - |

**Table 3: Benchmark statistics, displacement and runtime comparison on modified ISPD2015 benchmarks [11].**

| Case | #Cells of Different Heights ($H$) | | Den. (%) | Total Disp. (sites) | | | | | Runtime (s) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | | [11] | [8] | [6] | LEAGLM-CPU | LEAGLM-GPU | [11] | [8] | [6]† | LEAGLM-CPU | LEAGLM-GPU |
| mgc_superblue11_a | 861314 | 64302 | 43 | 1786342 | 1550976 | 1438304 | 1390481 | 1424208 | 21.16 | 28.91 | 99.43 | 500.33 | 53.46 |
| mgc_superblue12 | 1172586 | 114362 | 45 | 2015678 | 1726472 | 1615657 | 1533497 | 1562654 | 98.78 | 45.11 | 110.25 | 100.93 | 22.19 |
| mgc_superblue14 | 564769 | 47474 | 56 | 1599810 | 1359615 | 1159759 | 1112588 | 1142576 | 15.18 | 21.73 | 70.55 | 74.05 | 5.53 |
| mgc_superblue16_a | 625419 | 47474 | 48 | 1173106 | 1037827 | 946922 | 928873 | 955012 | 19.12 | 25.7 | 37.34 | 34.86 | 5.01 |
| mgc_superblue19 | 478109 | 27988 | 52 | 806529 | 694879 | 644336 | 640308 | 639796 | 9.68 | 17.83 | 27.39 | 17.45 | 3.12 |
| Ratio | - | - | - | 1.2853 | 1.1106 | 1.0114 | 0.9809 | 1.0000 | 2.25 | 2.63 | 5.99 | 7.40 | 1.00 |

† The runtime data is quoted from the publications for reference.

that the ALM-based legalization takes around 39% of the overall runtime. The initial legalization takes 3% of the runtime running on CPU only. CPU-GPU data transfer and file I/O take around 7% and 25% of the runtime, respectively. In practice, we can initialize all data in the GPU global memory before running the entire placement flow, thus this overhead is not mandatory.

## 5 Conclusion

In this work, we introduce LEGALM, an innovative mixed-cell-height legalization algorithm with routability and fence region constraints. We propose an augmented Lagrangian formulation coupled with a linearized proximal gradient descent method that offers a novel analytical perspective on cell position updates. We propose a block gradient descent method that enhances parallelism while maintaining the convergence speed of the augmented Lagrangian method. We present a triplefold partitioning technique that maximizes the utilization of GPU parallel computing power. Extensive experiments conducted on ICCAD–2017 benchmarks and modified ISPD–2015 benchmarks demonstrate LEGALM's superior performance compared to existing state-of-the-art legalizers in terms of both solution quality and runtime efficiency. We achieve a 6 – 36% improvement in overall quality score when compared to state-of-the-art legalizers on ICCAD–2017 benchmarks. On larger-scale designs with millions of cells, our proposed method shows even more significant acceleration effects, achieving a 2.25 – 5.99× speedup compared to other state-of-the-art methods. Our ablation studies further validate the efficacy of the triplefold partitioning technique, revealing up to 94.2× speedup with less than 0.5% quality score degradation. Future work includes exploring advanced parallelization strategies and integrating machine learning techniques to dynamically adjust partitioning schemes based on specific layout characteristics.

## Acknowledgments

This work was supported by National Science Foundation of China (Grant No. T2293700, T2293701), the Natural Science Foundation of Beijing, China (Grant No. Z230002), and 111 project (B18001).

## References


[1] Charles J. Alpert, Dinesh P. Mehta, and Sachin S. Sapatnekar. 2008. *Handbook of Algorithms for Physical Design Automation.* CRC press.

[2] Dimitri P. Bertsekas. 1999. *Nonlinear programming.* Athena scientific Belmont.

[3] Dimitri P Bertsekas. 2014. *Constrained optimization and Lagrange multiplier methods.* Academic press.

[4] Ulrich Brenner. 2013. BonnPlace Legalization: Minimizing Movement by Iterative Augmentation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 32, 8 (2013), 1215–1227.

[5] Ismail S. Bustany, David Chinnery, Joseph R. Shinnerl, and Vladimir Yutsis. 2015. ISPD 2015 benchmarks with fence regions and routing blockages for detailed-routing-driven placement. In *ACM International Symposium on Physical Design (ISPD)*. 157–164.

[6] Jinwei Chen, Zhixiong Di, Jiangyi Shi, Quanyuan Feng, and Qiang Wu. 2021. NBLG: A Robust Legalizer for Mixed-Cell-Height Modern Design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 41, 11 (2021), 4681–4693.

[7] Jianli Chen, Ziran Zhu, Wenxing Zhu, and Yao-Wen Chang. 2017. Toward Optimal Legalization for Mixed-Cell-Height Circuit Designs. In *ACM/IEEE Design Automation Conference (DAC)*. 52:1–52:6.

[8] Jianli Chen, Ziran Zhu, Wenxing Zhu, and Chang Yao-Wen. 2020. A Robust Modulus-Based Matrix Splitting Iteration Method for Mixed-Cell-Height Circuit Legalization. *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 26, 2 (2020).

[9] C. Cheng, A. B. Kahng, I. Kang, and L. Wang. 2019. RePlAce: Advancing Solution Quality and Routability Validation in Global Placement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 38, 9 (2019), 1717–1730.

[10] Minsik Cho, Haoxing Ren, Hua Xiang, and Ruchir Puri. 2010. History-based VLSI legalization using network flow. In *ACM/IEEE Design Automation Conference (DAC)*. 286–291.

[11] Wing-Kai Chow, Chak-Wa Pui, and Evangeline F. Y. Young. 2016. Legalization Algorithm for Multiple-Row Height Standard Cell Design. In *ACM/IEEE Design Automation Conference (DAC)*. 83:1–83:6.

[12] Nima Karimpour Darav, Ismail S. Bustany, Andrew Kennings, and Ravi Mamidi. 2017. ICCAD-2017 CAD Contest in Multi-Deck Standard Cell Legalization and Benchmarks. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 867–871.

[13] Nima Karimpour Darav, Ismail S. Bustany, Andrew Kennings, David Westwick, and Laleh Behjat. 2018. Eh?Legalizer: A High Performance Standard-Cell Legalizer Observing Technology Constraints. 23, 4 (2018).

[14] Dwight Hill. 2002. Method and system for high speed detailed placement of cells within an integrated circuit design. US Patent 6,370,673.

[15] Chung-Yao Hung, Peng-Yi Chou, and Wai-Kei Mak. 2017. Mixed-Cell-Height Standard Cell Placement Legalization. In *ACM Great Lakes Symposium on VLSI (GLSVLSI)*. 149–154.

[16] Nima Karimpour Darav, Ismail S Bustany, Andrew Kennings, and Laleh Behjat. 2017. A fast, robust network flow-based standard-cell legalization method for minimizing maximum movement. In *ACM International Symposium on Physical Design (ISPD)*. 141–148.

[17] Haocheng Li, Wing-Kai Chow, Gengjie Chen, Evangeline F. Y. Young, and Bei Yu. 2018. Routability-Driven and Fence-Aware Legalization for Mixed-Cell-Height Circuits. In *ACM/IEEE Design Automation Conference (DAC)*. 1–6.

[18] Xingquan Li, Jianli Chen, Wenxing Zhu, and Yao-Wen Chang. 2019. Analytical Mixed-Cell-Height Legalization Considering Average and Maximum Movement Minimization. In *ACM International Symposium on Physical Design (ISPD)*. 27–34.

[19] Yibo Lin, Shounak Dhar, Wuxi Li, Haoxing Ren, Brucek Khailany, and David Z. Pan. 2019. DREAMPlace: Deep Learning Toolkit-Enabled GPU Acceleration for Modern VLSI Placement. In *ACM/IEEE Design Automation Conference (DAC)*.

[20] Jingwei Lu, Pengwen Chen, Chin-Chih Chang, Lu Sha, Dennis Jen-Hsin Huang, Chin-Chi Teng, and Chung-Kuan Cheng. 2015. ePlace: Electrostatics-based placement using fast fourier transform and Nesterov's method. *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 20, 2 (2015), 1–34.

[21] Peter Spindler, Ulf Schlichtmann, and Frank M. Johannes. 2008. Abacus: fast legalization of standard cell circuits with minimal movement. In *ACM International Symposium on Physical Design (ISPD)*. 47–53.

[22] Chao-Hung Wang, Yen-Yi Wu, Jianli Chen, Yao-Wen Chang, Sy-Yen Kuo, Wenxing Zhu, and Genghua Fan. 2017. An effective legalization algorithm for mixed-cell-height standard cells. In *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*. 450–455.

[23] Haoyu Yang, Kit Fung, Yuxuan Zhao, Yibo Lin, and Bei Yu. 2022. Mixed-Cell-Height Legalization on CPU-GPU Heterogeneous Systems. In *IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE)*. 784–789.

[24] Vladimir Yutsis, Ismail S. Bustany, David Chinnery, Joseph R. Shinnerl, and Wen-Hao Liu. 2014. ISPD 2014 benchmarks with sub-45nm technology rules for detailed-routing-driven placement. In *ACM International Symposium on Physical Design (ISPD)*. 161–168.

[25] Ziran Zhu, Jianli Chen, Zheng Peng, Wenxing Zhu, and Yao-Wen Chang. 2018. Generalized augmented lagrangian and its applications to VLSI global placement. In *ACM/IEEE Design Automation Conference (DAC)*. 1–6.

[26] Ziran Zhu, Jianli Chen, Wenxing Zhu, and Yao-Wen Chang. 2020. Mixed-Cell-Height Legalization Considering Technology and Region Constraints. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 39, 12 (2020), 5128–5141.