GTA: GPU-Accelerated Track Assignment with Lightweight Lookup Table for Conflict Detection

Chunyuan Zhao[1], Jiarui Wang[1,2], Xun Jiang[1], Jincheng Lou[1], Yibo Lin[1,3,4*]
[1] School of Integrated Circuits, Peking University, Beijing, China
[2] School of Computer Science, Peking University, Beijing, China
[3] Institute of Electronic Design Automation, Peking University, Wuxi, China
[4] Beijing Advanced Innovation Center for Integrated Circuits
{zhaochunyuan, xunjiang, jinchenglou}@stu.pku.edu.cn, {jiaruiwang, yibolin}@pku.edu.cn

***Abstract*—Routing remains one of the most computationally intensive stages in VLSI physical design. Track assignment serves as a critical bridge between global routing (GR) and detailed routing (DR), offering more accurate routability estimation than GR while providing an initial solution for DR. However, existing approaches exhibit two key limitations: (1) Most track assignment methods are not aware of design rules, which makes they are unable to provide accurate congestion analysis and high-quality initial solution for detailed routing. (2) Current algorithms are exclusively designed for CPU architectures, which leads to limited parallelism and long runtime. This paper presents a novel GPU-accelerated track assignment framework that holistically addresses these design rules. Our implementation demonstrates significant improvements over the state-of-the-art detailed router TritonRoute-WXL, achieving 20× faster runtime and 25% reduced cpu peak memory usage, while maintaining competitive detailed routing quality. The proposed framework effectively bridges the gap between computational efficiency and design rule awareness in modern VLSI routing.**

I. Introduction

Routing plays a critical role in the design flow of modern very-large-scale-integrated (VLSI) circuits. With the increasing number of design rules and the complexities associated with advanced technology nodes, the routing solution is significantly influenced by these design rules, including the parallel run length spacing rule, corner-to-corner spacing rule, and cut spacing rule.

As shown in Figure 1, a typical routing flow takes the result after placement as its input. It performs global routing, track assignment and detailed routing. Global routing (GR) is typically used to obtain an approximate solution for routing resource utilization during the early design stages (i.e., placement). As shown in Figure 2(a), the typical global routing stage partitions the routing region into a set of *GCells* (global cells) and determines global routes for each signal net, aiming to minimize total wirelength, the number of vias, and overall congestion. The global routing solution serves as a guide for the detailed routing stage, providing a sound search region for each net. This stage is commonly employed to estimate routing resource utilization and design rule violation (DRV) distribution during early design phases, such as placement [1]–[3]. However, there is a considerable gap between the routing resource utilization predicted by global routing and that achieved by detailed routing. This discrepancy arises because global routing does not take specific design rules into account and relies solely on congestion as an estimate for potential design rule violations. Moreover, without a robust initial solution, detailed routers often struggle to find a high-quality routing solution within an affordable runtime, further exacerbating the challenge.

To bridge the gap between global routing and detailed routing, track assignment was introduced [4] as illustrated in Figure 1. This

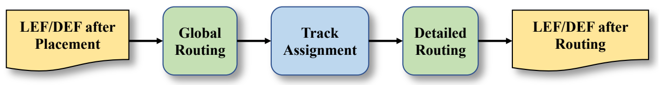


Fig. 1 A typical routing flow.

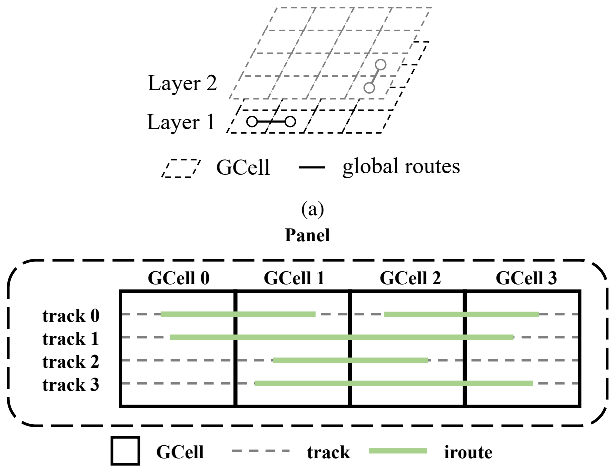


Fig. 2 (a) GCell grid and global routes, (b) panel, tracks and iroutes.

stage has proven to be an effective way to address specific design rules and other issues related to detailed routes, such as coupling effects, timing, and yield optimization [5]–[9]. Track assignment provides more accurate estimates of routing resource utilization and design rule violations at the early design stage and serves as a strong initial solution for detailed routing. As shown in Figure 2(a), we refer to long straight paths in the global routing solution as *iroutes* (intervals of global routes) and a full row or column (depends on the preferred direction) of GCells as *panel*. In a typical routing flow, we need to place most of the wire connections on *track*s. Figure 2(b) shows an example of a panel with 4 GCells, and there are 4 tracks to place iroutes in it. The track assignment stage assigns each iroute to a routing track within its corresponding panel, considering both local constraints and design rules. Figure 3 shows an example of a track assignment solution for two nets.

Track assignment can be roughly categorized into overlap aware track assignment and design rule aware track assignment. Overlap aware track assignment method only cares about wirelength, the overlapping length between an iroute and a blockage and the overlapping length between an iroute and another iroute. Many previous work have tried many methods to minimize the wirelength cost and

*Corresponding author

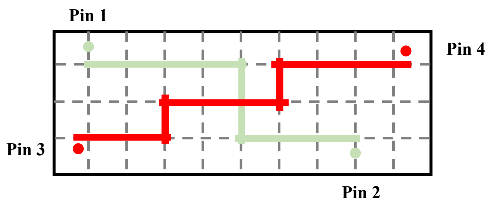


Fig. 3 An example of a track assignment solution for two nets (pin 1 $\rightarrow$ pin 2, pin 3 $\rightarrow$ pin 4).

overlapping length cost. [4] uses a weighted bipartite matching algorithm with a look heuristic strategy, achieving significantly reduced total routing time with acceptable quality degradation. [10] accounts for both global and local nets, using a negotiation-based approach to reduce the total cost. This negotiation-based strategy is widely used in other optimization problem such as global routing [11]–[14]. [15] incorporates more detailed connections between iroutes, focusing on via and pin connections to improve routability estimation. [16] uses a simulated annealing framework to solve the optimization problem. [17] frames the overlap aware track assignment as an integer linear programming (ILP) problem, solving it with an ILP solver and achieves less overlapping cost than methods above. It also applies a two-stage partition strategy to reduce the scale of the ILP problem with little quality degradation, but it still suffers from much longer runtime. Design rule aware assignment method is aware of specific design rules such as end-of-line spacing rule and cut spacing rule. Instead of overlapping cost, design rule violation cost matters. [18] considers via locations and routing resource utilization occupied by vias, providing a more accurate resource utilization estimate than commercial global routing and other academic track assignment frameworks. [19] represents the state-of-the-art academic router with a comprehensive global routing, track assignment, and detailed routing flow. They consider metal short rule, parallel run length spacing rule, and cut spacing rule during the track assignment stage, resulting in a high-quality detailed routing solution. However, these two kinds of track assignment have their own weakness respectively. Overlap aware track assignment is not aware of any specific design rule, which leads to a weak correlation with actual design rule violation and routing utilization in detailed routing. Design aware track assignment has a better correlation but it suffers from low runtime efficiency.

To accelerate electronic design automation (EDA) flow, GPU is introduced into many design stages due to its massive parallelism and high memory bandwidth. Many GPU-accelerated algorithms have demonstrated enhanced efficiency without sacrificing quality in various EDA applications [3], [20]–[27]. Historically, track assignment algorithms were designed for CPUs, which could only handle a limited number of threads on CPU platforms.

To handle a broader range of design rules and leverage GPU computational power, we propose `GTA`, a GPU-accelerated track assignment method that constructs a GPU-friendly lightweight lookup table and utilizes heterogeneous GPU computing platforms for fast, design rule-aware track assignment. This novel approach not only accelerates the track assignment process but also improves the quality of the initial solution for detailed routing. We summarize our main contributions as follows:

- We introduce a new GPU-friendly lightweight look-up table based on the GCell grid to enable efficient conflict detection, significantly improving runtime.
- We propose a maximal independent set-based batching strategy, which achieves significantly higher parallelism compared to naive inter-panel parallel strategies, leading to faster processing.
- To the best of our knowledge, we have developed the first GPU-accelerated track assignment framework which is highly scalable and can efficiently handle large designs. Compared to TritonRoute-WXL [28], our method achieves approximately 20× speedup in track assignment runtime, 25% CPU less peak memory usage. Meanwhile, our detailed routing quality is comparable or even better than [28].

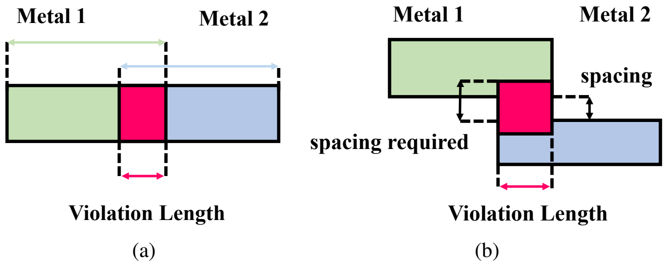


Fig. 4 Example of design rule violation lengths for (a) the metal short rule and (b) the parallel run length rule.

The remainder of this paper is organized as follows: Section II reviews the background and formulates the problem addressed in our work. Section III provides a detailed explanation of our proposed track assignment methods. Section IV validates our approach with comprehensive experimental results. Section V concludes the paper.

II. Preliminaries

In this section, we introduce the track assignment problem and provide an overview of the basic concepts involved in the process.

A. *Design Rule Checking*

Among the design rules defined in ISPD-2018 [29] and ISPD-2019 [30] benchmarks, we choose fundamental design rules addressed by our track assignment framework as follows:

- Metal short rule: A wire metal or via metal cannot overlap with any other metal object, such as pins, obstacles, wire metal, or via metal from other nets.
- End-of-line (EOL) spacing rule: A metal edge shorter than the defined *eolWidth* must maintain a spacing of at least *eolSpace* beyond the EOL to either side, with a distance less than *eolWithin*.
- Parallel run spacing rule: Two metal objects with a *parallelRunLength* must maintain a specified spacing, which is dependent on their *width* and *parallelRunLength*.
- Cut spacing rule: Two vias located on the same cut layer must maintain a spacing between each other.

In our track assignment formulation, we use the design rule violation length to quantify the extent to which a track assignment solution violates the design rules. The design rule violation length refers to the length of the region where design rule violations occur, measured along its preferred direction. Figure 4 provides an example of design rule violations for the metal short rule and parallel run length rule.

B. *Track Assignment Formulation*

As shown in Figure 2(a), in a 3D multi-layer design, the layout is partitioned into small rectangular regions known as GCells, defined by a grid of horizontal and vertical lines. During the global routing stage, the pins of signal nets are mapped to their corresponding GCells, and the global router generates a GCell-level routing solution, referred to as global routes. These global routes are also termed iroutes as described in previous work [4]. Due to manufacturing constraints, each metal layer has a preferred routing direction (either horizontal or vertical), and routing wires must follow this direction to ensure manufacturability. A contiguous row or column of GCells aligned with the preferred routing direction is referred to as a panel. Within each panel, a limited number of routing tracks are available for wire placement. The track assignment process determines the specific track for each global route, optimizing objectives such as minimizing wirelength and avoiding DRVs. The track assignment objective is thus formulated as:

$$\min \alpha \times WL + \beta \times DRVL + \gamma \times A \tag{1}$$

where WL represents the estimated total wirelength, $DRVL$ denotes the total length of design rule violations, such as metal overlap length, and A is an extra penalty term for detailed routing, which will be introduced in Section III.

C. Negotiation-based Track Assignment

The negotiation-based method is a widely used approach to solve routing and track assignment problems, and it has been proven to be both efficient and effective [10], [11], [13]–[15], [31], [32].

A typical negotiation-based method can be divided into two stages. The first stage, known as the initial routing or initial assignment, finds a initial solution. The second stage iteratively refines this solution to improve upon the initial assignment. In this refinement stage, the algorithm rips up routing paths with overflow or iroutes that violate design rules, and then it re-routes or re-assigns them with reduced overflow or fewer violations. In different iterations, the objective function assigns varying weights to overflow or violations, and the routing or assignment order may change to address different issues. Some studies also incorporate a history cost method during the refinement stage to help escape local optima.

Our proposed track assignment approach consists of three stages: database construction, initial assignment, and rip-up and re-assign. In the database construction stage, we build a lightweight lookup table to identify iroutes and blockages that may result in violations for a specific iroute. During the initial assignment stage, we assign each iroute primarily by considering wirelength, which provides an initial assignment solution. In the rip-up and re-assign stage, we select iroutes with violations and dynamically re-assign them in order to reduce the violations effectively.

III. Algorithm Framework

A. Overall Flow

Figure 5 illustrates the overall flow of our track assignment framework. The process begins with reading the pin access results and the global routing solution. Subsequently, we slice the guides and retrieve the pin access, the panel, the start GCell, and the end GCell for each iroute. Following this, we construct our proposed lightweight lookup table. Once these preprocessing is completed, our algorithm proceeds to the initial assignment stage, followed by the DRV reduction stage. We use the proposed unified assignment algorithm in Section 6 to do initial assignment and re-assignment. The main difference of initial assignment and rip-up and re-assignment lies in different weights α, β, γ in Equation 1 and different assignment order strategy, which will be explained in Section III-D. Finally, the track assignment results are passed to the detailed router to produce the final detailed routing outcome.

For better understanding of our framework, we first introduce our proposed lightweight lookup table construction in Section III-B. Then we explain the details of two algorithms powered by our

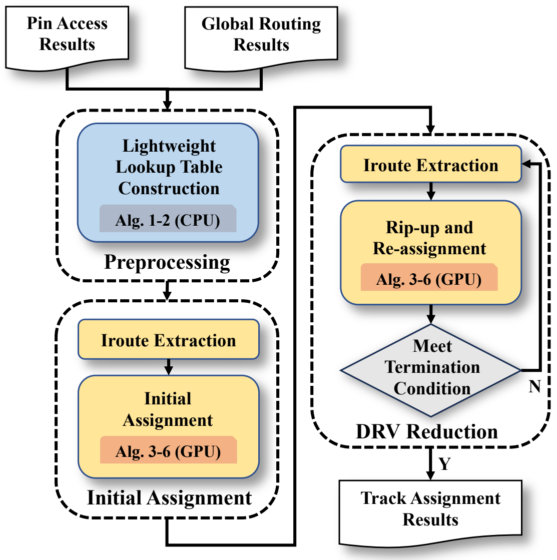


Fig. 5 The flow of our proposed GTA algorithm framework.

lightweight lookup table: the maximal independent set based batching strategy in Section III-C, and the unified assignment algorithm in Section III-D. Finally, we introduce some techniques

B. Lightweight Lookup Table for Conflict Detection

In this subsection, we present the construction of our lightweight lookup table, which is essential for DRV checking and for selecting batches for concurrent assignment.

For each iroute, we define its conflict set as the collection of iroutes that may violate design rules when assigned to the same tracks. Identifying the conflict set of an iroute is critical in track assignment, as it allows us to assess how many design rule violations an iroute may incur when assigned to different tracks. Furthermore, conflicting iroutes cannot be assigned simultaneously, as this could interfere with the convergence of the algorithm. R-tree [33] is a widely used data structure for spatial index queries in detailed routing and track assignment [28]. However, R-tree is not ideal for conflict detection in track assignment problems due to its long runtime and significant memory usage. To address this issue, we propose a lightweight lookup table to efficiently identify conflict sets, thereby reducing both runtime and memory consumption. Unlike R-tree, which performs poorly on GPUs, our lookup table is highly efficient and easily implemented on GPU architectures

For simplicity, we take the metal short rule as an example for illustration. In other words, the conflict set for an iroute consists of all other iroutes that share at least one common GCell with it. Our algorithm can be easily extended to handle additional design rules by increasing the size of the conflict region. In Figure 6, we have 5 iroutes within the same panel on the same metal layer : i_1 starts at G_1 and ends at G_3, i_2 starts at G_1 and ends at G_4, i_3 starts at G_2 and ends at G_3, i_4 starts at G_1 and ends at G_2 and i_5 starts at G_3 and ends at G_4. We take i_3 as an example and try to identify its conflict set. i_3 may have DRV with i_1 in $\{G_2, G_3\}$, i_2 in $\{G2, G3\}$, i_4 in $\{G_2\}$ and i_5 in $\{G_4\}$, respectively. Therefore, the conflict set of i_3 is $\{i_1, i_2, i_4, i_5\}$. These iroutes in the conflict set of i_3 can be classified into two categories: i_1, i_3 and i_5 have at least one endpoint in the GCells overlapped with i_3 ($\{G_2, G_3\}$). i_2 has no endpoint in the the GCells overlapped with i_3. Our proposed lookup table $M_{G,I}$ and $M_{I,I}$ are designed to identify conflict iroutes

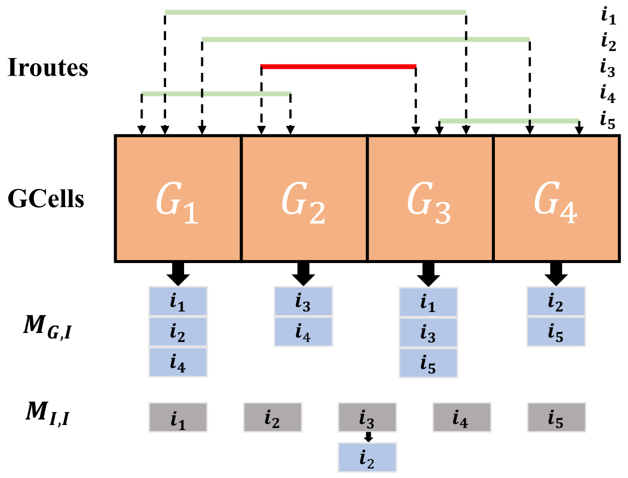


Fig. 6 An example of our lightweight lookup table.

of these two categories, respectively. We next explain how to build $M_{G,I}$ and $M_{I,I}$ and how they work in detail.

Let the panel in which iroute i runs be denoted as p_i of layer l_i. The start GCell of iroute i is denoted by G_{s_i}, and the end GCell of iroute i is denoted by G_{e_i}. It is easy to filter out iroutes in the same panel on the same layer, so we only talk about iroutes in the same panel on the same layer. For another iroute j, it will belong to the conflict set of iroute i if it satisfies the following Eq. (2):

$$s_i \le e_j \quad \text{and} \quad e_i \ge s_j \tag{2}$$

The Eq. (2) is satisfied if and only if Eq. (3a) or Eq. (3b) is satisfied:

$$s_i \le s_j \le e_i \quad \text{or} \quad s_i \le e_j \le e_i \tag{3a}$$
$$s_j < s_i \quad \text{and} \quad e_j > e_i \tag{3b}$$

Eq. (3a) describes that iroute j has at least one endpoint in the GCells overlapped with iroute i, and Eq. (3b) describes that iroute j has no endpoint in the GCells overlapped with iroute i. We construct the lightweight lookup tables $M_{G,I}$ to handle Eq. (3a), and $M_{I,I}$ to handle Eq. (3b), respectively.

As shown in Algorithm 1, we first construct the lookup table $M_{G,I}$ on the CPU. In this table, we store the iroutes that correspond to each GCell, where the GCell serves as either the starting or the ending point.

$$M_{G,I}\,(g) = \{i \mid G_{s_i} = g \text{ or } G_{e_i} = g\} \tag{4}$$

Then, using $M_{G,I}$, we construct the lookup table $M_{I,I}$ on the CPU. This second table stores the set of iroutes that satisfy Eq. (3b) for each iroute.

$$M_{I,I}\,(i) = \{j \mid s_j < s_i \text{ and } e_j > e_i\} \tag{5}$$

In general, for an iroute i that starts at G_s and ends at G_e, its conflict set is given by:

$$M_{G,I}(G_s) \cup M_{G,I}(G_{s+1}) \cup \cdots \cup M_{G,I}(G_e) \cup M_{I,I}(i) \setminus \{i\} \tag{6}$$

Figure 6 illustrates an example of our lightweight lookup table. We have five iroutes $(i_1, i_2, i_3, i_4, i_5)$ in the same panel with four GCells. Let $G = \{G_1, G_2, G_3, G_4\}$ and $I = \{i_1, i_2, i_3, i_4, i_5\}$. For example, i_1 starts at G_1 and ends at G_3, i_2 starts at G_1 and ends at G_4, and so on. We construct the lookup tables $M_{G,I}$ and $M_{I,I}$ for this scenario. $M_{G,I} = \{G_1 \rightarrow \{i_1, i_2, i_4\}, G_2 \rightarrow \{i_3, i_4\}, G_3 \rightarrow \{i_1, i_3, i_5\}, G_4 \rightarrow \{i_2, i_5\}\}$. $M_{I,I} = \{i_1 \rightarrow \emptyset, i_2 \rightarrow \emptyset, i_3 \rightarrow \{i_2\}, i_4 \rightarrow \emptyset, i_5 \rightarrow \emptyset\}$. Therefore, the conflict set of i_3 is

$$M_{G,I}\,(G_2) \cup M_{G,I}\,(G_3) \cup M_{I,I}\,(i_3) \setminus \{i_3\} \tag{7a}$$
$$= \{i_3, i_4\} \cup \{i_1, i_3, i_5\} \cup \{i_2\} \setminus \{i_3\} \tag{7b}$$
$$= \{i_1, i_2, i_4, i_5\} \tag{7c}$$

We will leverage these lookup tables to perform DRV checking efficiently. Note that our lookup table can be easily stored on GPU, and the DRV checking method can be implemented on the GPU for enhanced performance.

Algorithms 3 and 4 demonstrate how to perform DRV checking with our lightweight lookup table. We compute the DRVL cost between ir_1 and ir_2 for all possible positions of ir_2 in Algorithm 3. For simple design rules like the metal short rule, this DRV checking

Algorithm 1: Lookup Table $M_{G,I}$ Construction (CPU)

```
Input: set of GCells $G$, set of iroutes $I$
Output: lookup table from GCells to iroutes $M_{G,I}$
1 for $iroute_i \in I$ do
2     $s_i, e_i \leftarrow$ GetInfo($iroute_i$)
3     Add($M_{G,I}, G_{s_i}, iroute_i$)
4     if $s_i \neq e_i$ then
5         Add($M_{G,I}, G_{e_i}, iroute_i$)
6     end
7 end
```

Algorithm 2: Lookup Table $M_{I,I}$ Construction (CPU)

```
Input: set of GCells $G$, set of iroutes $I$, mapping table
       $M_{G,I}$
Output: lookup table from iroutes to iroutes $M_{I,I}$
1 for $iroute_i \in I$ do
2     $s_i, e_i \leftarrow$ GetInfo($iroute_i$)
3     for $id \leftarrow s_i$ to $e_i$ do
4         for $iroute_j \in M_{G,I}(G_{id})$ do
5             $s_j, e_j \leftarrow$ GetInfo($iroute_j$)
6             if $s_j = id$ and $e_j < e_i$ then
7                 Add($M_{I,I}, iroute_j, iroute_i$)
```

Algorithm 3: UpdateCT (GPU)

```
Input: iroute to apply solution $ir_1$, iroute to update cost
       table $ir_2$, cost table $CT$, set of candidate tracks $T$,
       set of design rule to check $R$, boolean variable
       $isAdd$
1 Function UpdateCT($ir_1, ir_2, CT, T, R, isAdd$):
2     for $t \in T$ do
3         for $rule \in R$ do
4             $cost_{DRV} \leftarrow$ GetDRVLCost($ir_1, ir_2, t, rule$)
5             if $isAdd$ = True then
6                 AddCost($CT, t, cost_{DRV}$)
7             end
8             if $isAdd$ = False then
9                 SubCost($CT, t, cost_{DRV}$)
10            end
11        end
12    end
```

Algorithm 4: Apply Solution (GPU)

```
Input: iroute to apply solution $ir$, lookup table $M_{G,I}, M_{I,I}$,
       cost table $CT$, set of design rule to check $R$, boolean
       variable $isAdd$ (True for assign and False for rip up)
Function ApplySolution($ir, M_{G,I}, M_{I,I}, CT, isAdd$):
    $s, e \leftarrow$ GetInfo($ir$)
    for $id \leftarrow s$ to $e$ do
        for $iroute_j \in M_{G,I}(G_{id})$ do
            $s_j, e_j, T \leftarrow$ GetInfo($iroute_j$)
            if $s_j = id$ or $s_j < s$ then
                UpdateCT($ir, iroute_j, CT, T, R, isAdd$)
            end
        end
    end
    for $iroute_j \in M_{I,I}(ir)$ do
        $T \leftarrow$ GetInfo($iroute_j$)
        UpdateCT($ir, iroute_j, CT, T, R, isAdd$)
    end
    return
```

is efficient and can be performed on the GPU. Algorithm 4 identifies the target iroutes ir_2 that are needed for DRV checking with ir_1. The conflict set searching algorithm is divided into two parts: Lines 3-8 in Algorithm 4 find the iroutes satisfying Eq. (3a), and Lines 9-11 identify the iroutes satisfying Eq. (3b). For more complex design rules like the parallel-run spacing rule, EOL spacing rule, and cut spacing rule, we only need to check these rules in `UpdateCT` and expand the conflict set searching region by one GCell.

C. Maximal Independent Set based Batching Strategy with Lightweight Lookup Table

A typical multi-threaded track assignment algorithm on a CPU relies on panel-level parallelism [10], [28]. It treats the assignment of iroutes in different panels as independent tasks, performing these assignments concurrently. However, this naive panel-level parallelism has two major issues:

- Inter-track design rules, such as the parallel-run spacing rule, prevent parallel assignment between adjacent panels.
- There are thousands of iroutes within a panel, making insufficient parallelism for GPU with tens of thousands of threads.

In practice, we observe that an optimal parallel strategy should follow two key principles:

- Iroutes that may have design rule violations should not be assigned concurrently.
- Iroutes with higher priority (based on iroute length or other criteria) should be assigned before those with lower priority.

These principles ensure that parallel algorithms converge effectively and yield high-quality results. In our proposed scheme, the assignment process is divided into several passes. In each pass, we select a batch of iroutes to assign and re-assign them concurrently.

To improve parallelism, we introduce a parallel maximal independent set (MIS)-based batching strategy inspired by Blelloch's algorithm [34]. This strategy selects a batch of iroutes to assign in a single pass. Algorithm 5 describes the selection process in detail. The core idea is to treat each iroute as a node, with an edge between two iroutes if they may violate a design rule when assigned to the same track. An independent set in the iroute graph ensures that no two iroutes that may have a design rule violation are assigned to the same track simultaneously. In practice, we do not explicitly construct

Algorithm 5: MIS-Based Selection (GPU)

```
Input: set of iroutes $I$, mapping table $M_{G,I}, M_{I,I}$, priority
       cost table $K$
Output: set of iroutes $S$ to assign in this pass
Function MISBasedSelection($I, M_{G,I}, M_{I,I}, K$):
    $S \leftarrow \emptyset$
    for each thread $0 \leq i < |I|$ do                ▷ selection kernel
        $iroute_i \leftarrow$ GetElement($I, i$)
        $flag_i \leftarrow$ True
        $s_i, e_i \leftarrow$ GetInfo($iroute_i$)
        $key_i \leftarrow K(iroute_i)$
        for $id \leftarrow s_i$ to $e_i$ do
            for $iroute_j \in M_{G,I}(G_{id})$ do
                $key_j \leftarrow K(iroute_j)$
                if $key_i < key_j$ then
                    $flag_i \leftarrow$ False
                end
            end
        end
        for $iroute_j \in M_{I,I}(iroute_i)$ do
            $key_j \leftarrow K(iroute_j)$
            if $key_i < key_j$ then
                $flag_i \leftarrow$ False
            end
        end
        if $flag_i = True$ then
            $S \leftarrow S \cup \{iroute_i\}$
            $I \leftarrow I \setminus \{iroute_i\}$
        end
    end
    return $S$
```

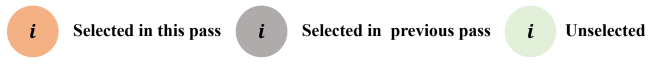
this graph, but instead, we implicitly access it using our lookup table. Lines 9-10 and 13-14 ensure that only the iroute $iroute_i$ with the highest priority among the iroutes in its conflict set is added to the set S. To satisfy the principles mentioned earlier, we select the maximal independent set, taking into account iroute priority, as the set of iroutes for re-assignment in each pass to ensure both efficiency and effectiveness.

Note that the selection task is fully parallelizable on a GPU, as each iroute can be processed independently. Since the priority value key_i of each iroute is updated after a pass of assignment, our method can achieve a higher degree of parallelism while maintaining dynamic ordering strategies. Compared with the naive inter-panel parallel strategy used in previous works, our method allows for intra-panel parallelism, improving both parallelism and efficiency.

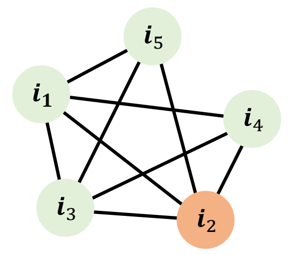
We illustrate our batching strategy using Figure 6 as an example. Suppose the iroute priority values satisfy $key_2 > key_1 > key_4 > key_5 > key_3$ for simplicity. In the first pass, i_2 is selected because it may conflict with i_1, i_3, i_4, i_5 and has the largest key value. We then remove i_2 from the candidate set I. In the second pass, i_1 is selected. In the third pass, i_4 and i_5 are selected, and in the last pass, i_3 is selected. With our approach, we can assign all 5 iroutes in the same panel in 4 passes, while a naive inter-panel parallel strategy would require 5 passes to assign them.

D. Unified Track Assignment Algorithm

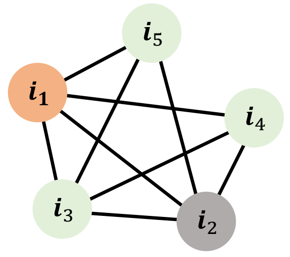
With the aforementioned components, we summarize our proposed unified track assignment algorithm on GPU in Algorithm 6



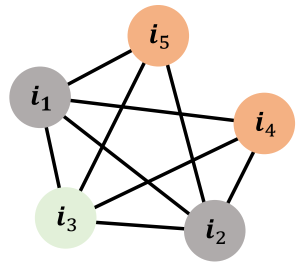
(a) Iroute selected in this / previous / later pass



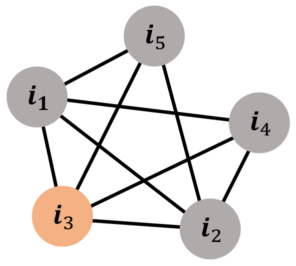
(b) Pass 1



(c) Pass 2



(d) Pass 3



(e) Pass 4

Fig. 7 An example of our proposed maximal independent set-based batching strategy using the iroutes from Figure 6.

that is general for initial assignment and rip-up and re-assignment. The algorithm runs iteratively and assign (or re-assign) a batch of iroutes in each iteration.

In each iteration of our track assignment, we attempt to assign (in the initial assignment iteration) or rip-up and re-assign (in the DRV reduction iteration) each iroute at most once. In each pass, we first select a batch of iroutes to assign concurrently, as described in Lines 2-7 of Algorithm 6. We use various ordering strategies to find the best solution, aiming to minimize the cost. In the initial assignment stage, the iroute length is used as the priority key for each iroute, while in the DRV reduction stage, the iroute DRVL cost is used as the priority key. We then launch the assign kernel on the GPU to assign a track to each iroute in the selected batch (Lines 8-19). Lines 11-13 retrieve the basic information for the iroute to be assigned, and Lines 14-19 find the best track with the minimal cost. Finally, we store the solution of each iroute and update the cost table (Lines 20-21). The entire process is fully parallelized on the GPU, with each thread assigned to one iroute. The initial assignment process continues until every iroute has been assigned. The rip-up and re-assignment process continues until no iroutes with DRVL violations remain unassigned in the current iteration. For the initial assignment, we set $\alpha = 1$, $\beta = 0.05$, and $\gamma = 0.05$. In the rip-up and re-assignment stage, we set $\alpha = 1$, $\beta = 32$, and $\gamma = 32$ for further DRV reduction.

E. Detailed Routing Consideration

To improve the quality of results after detailed routing, we need the objective function (1) has a good correlation with the detail routing result. In the subsection, we will explain the wirelength cost WL first, and then explain the details of the extra penalty A.

We define the iroutes that belong to the same net on adjacent layers and are linked to a particular iroute as its neighboring iroutes. An iroute is considered to be connected to its neighbors through vias.

Algorithm 6: Unified Track Assignment (GPU)

```
Input: set of GCells $G$, set of iroutes $I$, mapping table
       $M_{G,I}, M_{I,I}$, cost table $CT$
Output: track assignment solution
while True do
    if doing initial assignment then
        $S \leftarrow$ MISBasedSelection($I, M_{G,I}, M_{I,I}, L$)
            $\triangleright$ Use the length of iroutes as the priority cost
    end
    if doing re-assignment then
        $S \leftarrow$ MISBasedSelection($I, M_{G,I}, M_{I,I}, CT$)
            $\triangleright$ Use DRVL cost of iroutes as the priority cost
    end
    if $S = \emptyset$ then
        return
    end
    for each thread $0 \le i < |S|$ do            $\triangleright$ assign kernel
        $iroute_i \leftarrow$ GetElement($S, i$)
        if doing re-assignment then
            ApplySolution($iroute_i, M_{G,I}, M_{I,I}, CT$, False)
              $\triangleright$ Apply the cost update of ripping up $iroute_i$
        end
        $l_i, p_i \leftarrow$ GetInfo($iroute_i$)
        $T_i \leftarrow$ GetTracks($l_i, p_i$)
        $minCost \leftarrow \infty$
        for $t \in T_i$ do
            $cost_t \leftarrow$ GetCost ($t, CT_i$)
            if $cost_t < minCost$ then
                $minCost \leftarrow cost_t$
                $bestTrack \leftarrow t$
            end
        end
        SetSolution($iroute_i, bestTrack$)
        ApplySolution($iroute_i, M_{G,I}, M_{I,I}, CT$, True)
          $\triangleright$ Apply the cost update of assignment of $iroute_i$
    end
end
```

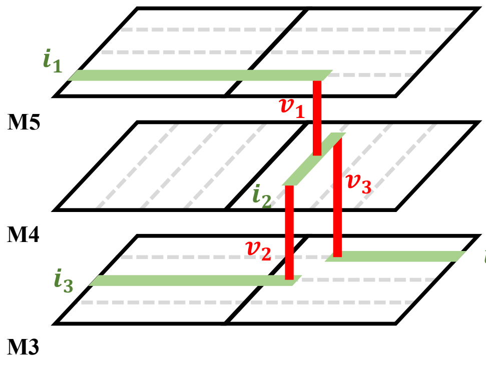
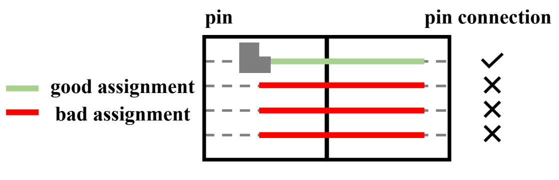
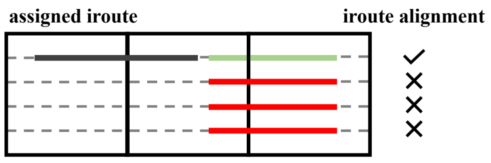


Fig. 8 Iroute and its neighbors.

These neighboring iroutes are used to estimate the total wirelength in detailed routing result. For example, in Figure 8, i_1 and i_3 and i_4 are the neighbors of i_2. Notably, we use relative wirelength cost instead of actual wirelength cost in the objective Eq. (1) because they share the same assignment solution. When i_2 is assigned to t_1, we gives 0 as its wirelength cost in the objective Eq. (1). i_1 and i_3



(a)



(b)

Fig. 9 (a) Pin connection and (b) iroute alignment.

TABLE I Benchmarks statistics.

ISPD18	#nets	GCell Grid	ISPD19	#nets	GCell Grid
test1	3153	67 × 68	test1	3153	97 × 98
test2	36834	201 × 228	test2	72410	392 × 581
test3	36700	247 × 346	test3	8953	130 × 130
test4	72401	403 × 592	test4	151612	518 × 534
test5	72394	613 × 619	test5	29416	302 × 302
test6	107701	354 × 571	test6	179863	883 × 905
test7	179863	883 × 905	test7	358720	1001 × 1053
test8	179863	883 × 905	test8	537577	1138 × 1202
test9	178857	522 × 606	test9	895253	1433 × 1337
test10	182000	522 × 606	test10	895253	1433 × 1337

comes from the left side on M5 and M3, respectively, while only i_4 comes from the right side on M3. When we move i_2 from track t_1 to track t_2, we have to lengthen i_1 and i_3 by one track step s, and shorten i_4 by s, and our total wirelength get s more. To describe the wirelength difference, we gives s and $2s$ as the wirelength cost of i_2 assigned to track t_2 and t_3, respectively.

To further improve the quality of detailed routing results, we introduce an extra penalty to reward assignment strategies that benefit detailed routing. Pin connection strategy means that iroutes are encouraged to be assigned close to their respective pins because it will prevent make detours to connect pins and it is good for routability. Figure 9(a) demonstrates an example of pin connection. The pin is located at the first track, so assigning the corresponding iroute to the first track is a pin connection pattern and assigning it to other tracks is not a pin connection pattern. Therefore, we give the solution of the first track a negative extra penalty to encourage our framework to assign it to the first track and achieve pin connection. Iroute alignment strategy means that an iroute is encouraged to be assigned to same track as its directly connected iroutes in the same net. Misaligned iroutes may require additional vias or wires to achieve connection and harm routability. Figure 9(b) shows an example of iroute alignment. The assigned iroute in the same net is located at the first track, so assigning the corresponding iroute to the first track satisfies iroute alignment. In our algorithm implementation, we also assign a negative extra penalty to encourage iroute alignment.

IV. Experimental Results

The benchmarks used in our experiments are sourced from the ISPD-2018 and ISPD-2019 routing contests. Table I provides detailed information about these benchmarks.

We implemented our track assignment using `C++`, `CUDA`, and `OpenMP` [35]. To align with ISPD-2018 [29] and ISPD-2019 [30] contest, the CPU utilized 8 threads during the experiments. We integrated our framework into TritonRoute-WXL [28] to do global routing and detailed routing, and compare our method with the original TritonRoute-WXL flow. We will first present our main comparison with TritonRoute-WXL and then we will show the ablation studies on our proposed lookup table and batching strategy. All experiments were conducted on a Linux machine equipped with a 64-core AMD EPYC 7542 32-Core Processor running at 2.90 GHz and an NVIDIA 3090 GPU.

A. Main Comparison with TritonRoute-WXL

We compare wirelength, via count, DRV count, runtime of track assignment, runtime of track assignment and detailed routing, and the peak memory usage of TritonRoute-WXL with our proposed framework in Table II. Our method achieves approimately a 20× speedup in track assignment runtime and a 3.7% speedup in the overall runtime of track assignment and detailed routing. In particular, for large cases (ispd19_test6 to ispd19_test10), our method simultaneously achieves better wirelength, fewer vias, fewer DRVs, shorter runtime, and lower peak memory usage.

Quality improvements arise from our more accurate estimation model and appropriate weights selection. These provide the detailed router with good initial solution with flexible enough search space. The runtime improvement stems from our fast DRV detection method and our maximal independent set based batching strategy with higher parallelism. Lower peak memory usage is achieved through our highly sparse and lightweight lookup table implementation.

Figure 10 shows a comparison of the runtime and peak memory usage between TritonRoute-WXL and our method as the number of nets increases. Our framework demonstrates superior scalability with the growing problem size.

B. Ablation Study

To evaluate the effectiveness of our proposed lightweight lookup table, we selected the three largest designs from the ISPD-2019 benchmark suite and collected the statistics of $|G|, |I|, |M_{G,I}|$ and $|M_{I,I}|$. As shown in Table III, for these designs, we observe that $|M_{G,I}| \approx |G|$ and $|M_{I,I}| \approx |I|$. This relationship allows us to construct our lookup table with linear space complexity. Furthermore, when querying the conflict set of an iroute which spans L GCells, our lookup table requires only $O(L)$ steps. We observe that a high-quality placer tends to place cells related to the same net in close proximity. Meanwhile, an effective global router aims to minimize wirelength and avoid *overflow* (demand exceeds capacity) as much as possible. As a result, the total length of global routes and the number of global routes passing through the same GCell are inherently limited. Due to these issues, our lookup table is sparse enough to provide lightweight conflict detection.

To demonstrate the effectiveness of our novel maximal independent set based batching strategy, we compare our method with the naive inter-panel parallel strategy used in prior works. The evaluation is conducted on the ten largest designs from the ISPD-2018 and ISPD-2019 benchmark suites. As shown in Table IV, our algorithm reduces the number of passes by approximately 48 times on average compared to previous methods. This indicates that our batching strategy can distribute all assignment tasks into 48 times fewer batches on average, significantly improving workload balance and reducing the maximal workload per single thread. This improvement is particularly critical and effective in our GPU-accelerated framework, as the computing power of a single GPU

TABLE II Comparison between TritonRoute-WXL and our framework for routed wirelength (rWL/um), routed via count (#rVia), routed DRV count (#rDRV), track assignemnt runtime (TA/s), track assignment + detailed routing runtime (TA+DR/s), cpu peak memory usage (CMem/MB) and gpu peak memory usage (GMem/MB).

Design[†]	TritonRoute-WXL [28]						Ours						
	rWL	#rVia	#rDRV	TA	TA+DR	CMem	rWL	#rVia	#rDRV	TA	TA+DR	CMem	GMem
18-1	85473	36081	**0**	1.69	15.71	**782**	**85448**	**36048**	**0**	**0.09**	**13.8**	864	28
18-2	**1561031**	**376987**	**0**	9.56	**118.98**	1701	1561129	377026	**0**	**0.55**	124.84	**1658**	234
18-3	**1748109**	**378138**	**0**	10.12	648.4	1811	1748736	378747	**0**	**0.62**	**420.62**	**1748**	236
18-4	**2608328**	754787	**0**	16.57	235.17	3681	2608669	**752847**	**0**	**0.84**	**219.82**	**2655**	452
18-5	2739659	927176	**0**	26.67	305.08	3626	**2739610**	**925719**	**0**	**1.16**	**253.26**	**2940**	504
18-6	**3517795**	1396704	**0**	31.39	412.5	4623	3518040	**1395092**	**0**	**1.54**	**398.48**	**3812**	706
18-7	**6419471**	2282189	**0**	52.04	956.45	6541	6419961	**2279871**	**0**	**2.59**	**903.08**	**5419**	1206
18-8	**6450850**	2362426	**0**	53.83	1048.03	6286	6451131	**2360074**	**0**	**2.64**	**935.1**	**5539**	1230
18-9	5387863	2343511	**0**	54.42	728.26	6527	**5387732**	**2341826**	**0**	**2.54**	**679.89**	**5344**	1174
18-10	**6826642**	2565687	**0**	70.36	1098.66	7458	6826799	**2563931**	**0**	**2.88**	**1085.88**	**5832**	1320
19-1	**62897**	**38495**	**0**	2.88	**38.61**	**831**	62934	38516	**0**	**0.11**	41.85	961	62
19-2	**2460494**	**864082**	**0**	27.11	619.84	3502	2460673	865270	**0**	**1.28**	**619.7**	**3030**	878
19-3	82227	64036	**0**	3.36	194.02	**1019**	**82202**	**63953**	**0**	**0.10**	**153.85**	1084	78
19-4	6811716	1176847	3	62.25	**1981.41**	3853	**6807715**	**1174955**	**1**	**2.13**	2041.62	**3374**	462
19-5	982248	154287	**0**	4.69	**61.62**	**920**	**981781**	**153625**	**0**	**0.24**	73.86	960	76
19-6	6513144	2060921	**0**	58.48	1296.86	7423	**6512618**	**2056949**	**0**	**3.27**	**1245.45**	**5571**	2226
19-7	12042993	3896349	**0**	119.18	2827.18	12130	**12042138**	**3892269**	**0**	**6.55**	**2740.74**	**8969**	4626
19-8	18494009	6425825	**0**	190.82	3324.28	16987	**18492141**	**6417225**	**0**	**9.97**	**3227.28**	**13600**	6652
19-9	27964340	10673288	**0**	325.11	5460.91	28396	**27962026**	**10660709**	**0**	**16.26**	**5325.43**	**21569**	11250
19-10	27607518	10038100	**4**	330.99	6357.13	26866	**27605735**	**10024519**	**4**	**15.73**	**6238.65**	**20985**	11300
Average	7018340	2440796	0.35	72.58	1386.46	7253.15	**7017860**	**2437959**	**0.25**	**3.55**	**1337.16**	**5795.7**	2235.0
Avg. ratio	**1.000**	1.001	1.400	20.418	1.037	1.251	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	-

[†] Design name 18-**x** and 19-**x** denote ISPD18-test**x** and ISPD19-test**x**, respectively.

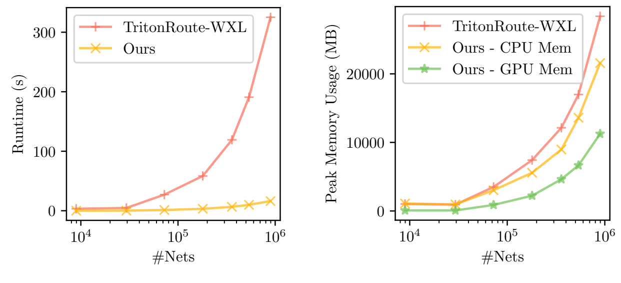
TABLE III $|G|, |I|, |M_{G,I}|, |M_{I,I}|, |M_{G,I}|/|G|$ and $|M_{I,I}|/|I|$ of three largest ISPD-2019 testcases.

	$\|G\|$	$\|I\|$	$\|M_{G,I}\|$	$\|M_{I,I}\|$	$\|M_{G,I}\|/\|G\|$	$\|M_{I,I}\|/\|I\|$
19-8	12310884	7152465	9883197	3984174	0.80	0.56
19-9	17243289	11855540	16368264	6501760	0.95	0.55
19-10	17243289	11152369	15602001	6430225	0.90	0.58

TABLE IV Comparison between panel parallelism and maximal independent set based parallelism.

Design		18-6	18-7	18-8	18-9	18-10	19-6	19-7	19-8	19-9	19-10	Average
#iroute		1561781	2540095	2611071	2591612	2807037	2309206	4294027	7152465	11855540	11152369	4887520
#pass	panel	2844	3597	3701	3518	3813	3945	5326	6221	7898	8237	4910
	MIS	121	115	106	90	104	127	108	88	104	97	106
#iroute / #pass	panel	549	706	705	737	736	585	806	1150	1501	1354	883
	MIS	12907	22088	24633	28796	26991	18183	39760	81278	113996	114973	48360

thread is limited. By minimizing the workload imbalance, our approach enhances the overall efficiency of the framework.



(a) (b)

Fig. 10 (a) Runtime and (b) peak memory usage scaling with the number of nets.

V. Conclusion

In this paper, we present a track assignment framework that generates high-quality track assignment solutions efficiently on the ISPD-2018 and ISPD-2019 benchmark suites. We introduce a new GPU-friendly, lightweight lookup table that improves the time and space complexity of conflict set detection for each iroute, outperforming traditional R-tree methods. This data structure enables efficient handling of various design rules concurrently on the GPU. Additionally, we employ a maximal independent set-based batching strategy to generate batches of iroutes for simultaneous assignment, ensuring intra-panel parallelism and contributing to significant speed-up in the track assignment process. Compared to the state-of-the-art router, TritonRoute-WXL, our method achieves competitive routing solutions, delivering a 20× speedup in track assignment runtime, and a 25% reduction in CPU peak memory usage.

Acknowledgements

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

References

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

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

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

[4] S. Batterywala, N. Shenoy, W. Nicholls, and H. Zhou, "Track assignment: a desirable intermediate step between global routing and detailed routing," in *IEEE/ACM International Conference on Computer Aided Design, 2002. ICCAD 2002.*, 2002, pp. 59–66.

[5] D. Wu, R. Mahapatra, J. Hu, and M. Zhao, "Timing driven track routing considering coupling capacitance," in *Proceedings of the ASP-DAC 2005. Asia and South Pacific Design Automation Conference, 2005.*, vol. 2, 2005, pp. 1156–1159 Vol. 2.

[6] M. Cho, H. Xiang, R. Puri, and D. Z. Pan, "Track routing and optimization for yield," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 5, pp. 872–882, 2008.

[7] Z. Qi, Q. Zhou, Y. Jia, Y. Cai, Z. Li, and H. Yao, "A novel fine-grain track routing approach for routability and crosstalk optimization," in *2011 12th International Symposium on Quality Electronic Design*, 2011, pp. 1–6.

[8] X. Gao and L. Macchiarlo, "Track routing optimizing timing and yield," in *16th Asia and South Pacific Design Automation Conference (ASP-DAC 2011)*, 2011, pp. 627–632.

[9] B.-T. Lai, T.-H. Li, and T.-C. Chen, "Native-conflict-avoiding track routing for double patterning technology," in *2012 IEEE International SOC Conference*, 2012, pp. 381–386.

[10] M.-P. Wong, W.-H. Liu, and T.-C. Wang, "Negotiation-based track assignment considering local nets," in *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2016, pp. 378–383.

[11] L. McMurchie and C. Ebeling, "Pathfinder: A negotiation-based performance-driven router for fpgas," in *Third International ACM Symposium on Field-Programmable Gate Arrays*, 1995, pp. 111–117.

[12] J.-R. Gao, P.-C. Wu, and T.-C. Wang, "A new global router for modern designs," in *2008 Asia and South Pacific Design Automation Conference*, 2008, pp. 232–237.

[13] Y. Xu, Y. Zhang, and C. Chu, "Fastroute 4.0: Global router with efficient via minimization," in *Proceedings of the 2009 Asia and South Pacific Design Automation Conference*, ser. ASP-DAC '09. IEEE Press, 2009, p. 576–581.

[14] J. He, U. Agarwal, Y. Yang, R. Manohar, and K. Pingali, "Sproute 2.0: A detailed-routability-driven deterministic parallel global router with soft capacity," in *2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2022, pp. 586–591.

[15] G. Liu, Z. Zhuang, W. Guo, and T.-C. Wang, "Rdta: An efficient routability-driven track assignment algorithm," in *Proceedings of the 2019 Great Lakes Symposium on VLSI*, ser. GLSVLSI '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 315–318. [Online]. Available: https://doi.org/10.1145/3299874.3318026

[16] Y. Qi, Z. Gan, J. Ding, Z. Fu, M. Gong, and W. Yu, "Track assignment using gradient indication and simulated annealing," in *2024 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2024, pp. 1–5.

[17] Y. Jing, L. Yang, Z. Zhuang, G. Liu, X. Huang, W.-H. Liu, and T.-C. Wang, "Spta: A scalable parallel ilp-based track assignment algorithm with two-stage partition," in *2022 IFIP/IEEE 30th International Conference on Very Large Scale Integration (VLSI-SoC)*, 2022, pp. 1–6.

[18] D. Shi and A. Davoodi, "Trapl: Track planning of local congestion for global routing," in *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2017, pp. 1–6.

[19] A. B. Kahng, L. Wang, and B. Xu, "Tritonroute: The open-source detailed router," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 3, pp. 547–559, 2021.

[20] Z. Guo, T.-W. Huang, and Y. Lin, "Heterocppr: Accelerating common path pessimism removal with heterogeneous cpu-gpu parallelism," in *2021 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. ACM, 2021.

[21] S. Lin, J. Liu, T. Liu, M. D. F. Wong, and E. F. Y. Young, "Novelrewrite: node-level parallel aig rewriting," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, ser. DAC '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 427–432. [Online]. Available: https://doi.org/10.1145/3489517.3530462

[22] Z. Guo, F. Gu, and Y. Lin, "Gpu-accelerated rectilinear steiner tree generation," in *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, ser. ICCAD '22. New York, NY, USA: Association for Computing Machinery, 2022. [Online]. Available: https://doi.org/10.1145/3508352.3549434

[23] S. Lin, J. Liu, E. F. Y. Young, and M. D. F. Wong, "Gamer: Gpu-accelerated maze routing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 2, pp. 583–593, 2023.

[24] S. Liu, Y. Pu, P. Liao, H. Wu, R. Zhang, Z. Chen, W. Lv, Y. Lin, and B. Yu, "Fastgr: Global routing on cpu–gpu with heterogeneous task graph scheduler," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 7, pp. 2317–2330, 2023.

[25] C. Zhao, Z. Guo, R. Wang, Z. Wen, Y. Liang, and Y. Lin, "Helem-gr: Heterogeneous global routing with linearized exponential multiplier method," in *2024 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. ACM, 2024.

[26] Y. Du, Z. Guo, Y. Lin, R. Wang, and R. Huang, "Fusion of global placement and gate sizing with differentiable optimization," in *2024 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. ACM, 2024.

[27] J. Mai, C. Zhao, Z. Zhang, Z. Di, Y. Lin, R. Wang, and R. Huang, "Legalm: Efficient legalization for mixed-cell-height circuits with linearized augmented lagrangian method," in *Proceedings of the 2025 International Symposium on Physical Design*, ser. ISPD '25. New York, NY, USA: Association for Computing Machinery, 2025, p. 22–30. [Online]. Available: https://doi.org/10.1145/3698364.3705356

[28] A. B. Kahng, L. Wang, and B. Xu, "Tritonroute-wxl: The open-source router with integrated drc engine," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 4, pp. 1076–1089, 2022.

[29] S. Mantik, G. Posser, W.-K. Chow, Y. Ding, and W.-H. Liu, "Ispd 2018 initial detailed routing contest and benchmarks," in *Proceedings of the 2018 International Symposium on Physical Design*, ser. ISPD '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 140–143. [Online]. Available: https://doi.org/10.1145/3177540.3177562

[30] W.-H. Liu, S. Mantik, W.-K. Chow, Y. Ding, A. Farshidi, and G. Posser, "Ispd 2019 initial detailed routing contest and benchmark with advanced routing rules," in *Proceedings of the 2019 International Symposium on Physical Design*, ser. ISPD '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 147–151. [Online]. Available: https://doi.org/10.1145/3299902.3311067

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

[32] S.-Y. Han, W.-H. Liu, R. Ewetz, C.-K. Koh, K.-Y. Chao, and T.-C. Wang, "Delay-driven layer assignment for advanced technology nodes," in *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2017, pp. 456–462.

[33] A. Guttman, "R-trees: a dynamic index structure for spatial searching," in *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '84. New York, NY, USA: Association for Computing Machinery, 1984, p. 47–57. [Online]. Available: https://doi.org/10.1145/602259.602266

[34] G. E. Blelloch, J. T. Fineman, and J. Shun, "Greedy sequential maximal independent set and matching are parallel on average," in *Proceedings of the Twenty-Fourth Annual ACM Symposium on Parallelism in Algorithms and Architectures*, ser. SPAA '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 308–317. [Online]. Available: https://doi.org/10.1145/2312005.2312058

[35] "OpenMP," http://www.openmp.org/.