HeLEM-GR: Heterogeneous Global Routing with Linearized Exponential Multiplier Method

Chunyuan Zhao[1], Zizheng Guo[1,2], Rui Wang[3], Zaiwen Wen[4], Yun Liang[1,2,5], Yibo Lin[1,2,5]
[1] School of Integrated Circuits, Peking University [2] Institute of EDA, Peking University
[3] School of Mathematics, Southwestern University of Finance and Economics
[4] Beijing International Center for Mathematical Research, Peking University
[5] Beijing Advanced Innovation Center for Integrated Circuits
zhaochunyuan@stu.pku.edu.cn, {gzz,wenzw,ericlyun,yibolin}@pku.edu.cn, wangr@swufe.edu.cn

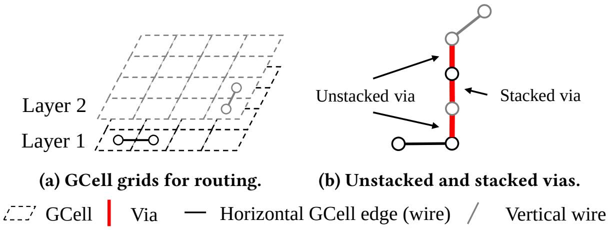
ABSTRACT

Global routing (GR) plays an important role in the VLSI design flow. It not only serves as guidance for the follow-up detailed routing but also provides early design feedback for floorplanning and placement. Global routing engines are desired to provide a high-quality solution within a short time. With the design complexity growing, it becomes increasingly challenging to resolve routing overflow within affordable runtime. For example, ISPD 2024 GPU/ML-enhanced global routing contest has released large-scale industrial cases, which contain up to 50 million cells and 60 million signal nets, causing huge challenges to existing routing algorithms. In this paper, we propose HeLEM-GR, based on the linearized exponential multiplier method and heterogeneous routing kernels to achieve high-quality and ultrafast routing solutions. Our linearized exponential multiplier method can quickly reduce routing overflow. The routing process is extremely fast with GPU-enhanced massive parallelization. Experimental results demonstrate that we can achieve 4.8%-5.8% better quality scores and 1.62×-2.07× speedup compared with the top-3 winners in the ISPD 2024 contest.

1 INTRODUCTION

Routing is critical to modern very-large-scale-integrated (VLSI) design flow. It decides the wire and via connection between different components in a circuit layout. As the design scale increases rapidly, routing becomes extremely time-consuming and hard to achieve design closure.

Routing is usually divided into global routing (GR) and detailed routing (DR) [2]. Global routing serves as a fast routing planning to generate guidance for detailed routing to reduce the search space of each net. Detailed routing then takes the guidance as input and finishes the physical wiring to connect pins in each net. Global routing is also used for routability estimation at early design stages like placement [3–9]. With growing design scales and complexity, it becomes increasingly challenging for global routing to resolve routing overflow within a short time. Therefore, the quality and efficiency of global routing is critical to design closure, as it impacts both its proceeding and succeeding design stages.



(a) GCell grids for routing. (b) Unstacked and stacked vias.

Figure 1: Illustration of the global routing problem [1].

The literature on global routing can be roughly categorized into three perspectives: grid models, routing kernels, and routing schemes. Mainly two types of grid models have been investigated, i.e., 2D grids and 3D grids. The 2D grid model projects 3D routing grids into 2D space. Many routers such as FastRoute 4.0 [10], BoxRouter 2.0 [11], NCTU-GR 2.0 [12], and SPRoute 2.0 [13] are based on 2D grids (a.k.a 2D routers), which perform layer assignment after routing all nets on the 2D space. Other routers such as FGR [14] and CUGR [15] try to directly route on 3D grids to simultaneously determine the routing paths and layers for each net. In general, 2D grids have much smaller search space to achieve efficient routing, while 3D grids can have a better view of the congestion at different layers. TritonRoute-WXL [16] uses a hybrid model, which generates initial routing results on 2D grids and then refines it on 3D grids.

Various studies have also explored techniques to speedup routing kernels for each net, including maze routing and pattern routing. Lee's algorithm [17] is the basic maze routing kernel in many routers, but it is very time-consuming. Follow-up studies try to prune the search space by limiting the bounding boxes [12], coarse-fine grids [15], etc. Pattern routing [12, 13, 15, 18] directly uses specific routing patterns like L shape and Z shape to quickly route those 'easy' nets. EDGE [19] proposes a DAG-based routing method to generalize different patterns by modifying the routing graph. Some studies have explored different CPU multi-threading strategies to perform the routing kernels for different nets concurrently [13, 15, 20]. A few studies also propose to accelerate routing kernels on heterogeneous computing resources like GPU and FPGA, including maze routing [21–24], and L/Z shape pattern routing [24, 25]. However, these pattern routing kernels cannot generate a good enough solution, and maze routing kernels are not efficient enough when facing large-scale global routing problems.

Given the grid models and routing kernels, most routers adopt the rip-up and reroute scheme to minimize overflow iteratively. Studies like [10, 12, 13, 26] integrate a history cost into a heuristic function to guide routing kernels to escape from congested regions. Other work like [27] tries to explore good routing orders to reduce overflow. As the heuristic cost function may not have

ICCAD '24, October 27–31, 2024, New York, NY, USA

ACM ISBN 979-8-4007-1077-3/24/10…$15.00
https://doi.org/10.1145/3676536.3676650

mathematical insights, recent studies such as [28–30] formulate mathematical programming models to derive how to update the cost function during the rip-up and reroute scheme. Some studies such as [31] formulate the GR problem as a multicommodity flow problem and get a routing solution by multicommodity flow algorithm. However, the aforementioned methods still encounter challenges like oscillation and slow convergence in resolving overflow due to the increasing complexity of modern designs.

To stimulate research on global routing, the ISPD 2024 GPU/ML-enhanced global routing contest held by NVIDIA has released large-scale industrial designs, containing up to 50 million cells and 60 million signal nets, which is extremely challenging for existing routing algorithms. To handle these large scale cases in an affordable time, we propose HeLEM-GR, which formulates a 2D routing algorithm based on the linearized exponential multiplier method and utilizes heterogeneous CPU-GPU platforms to achieve ultrafast routing. We summarize our main contributions as follows:

- We propose a linearized exponential multiplier (LEM for short) method for the 2D routing problem to minimize wirelength and overflow. This LEM framework is general to integrate any routing kernels.
- We propose well-optimized batched routing kernels including L shape and 3-bend routing for GPU parallelization. Our RMQ-based 3-bend routing algorithm on GPU improves the runtime complexity from linear to logarithmic and thus mitigates the workload imbalance issue.
- We propose a GPU-accelerated layer assignment algorithm based on the sweep operations by describing it as a shortest path problem on a directed grid graph.

Experimental results demonstrate that we can achieve 4.8%-5.8% better quality scores (considering wirelength, vias, and overflow) and 1.62×-2.07× speedup compared with the top-3 winners in the ISPD 2024 contest [1].

The rest of this paper is organized as follows. Section 2 reviews the background and formulates the problem in our work. Section 3 provides a thorough explanation of the proposed routing methods. Section 4 validates our approach with comprehensive experimental results. Section 5 concludes the paper.

2 PRELIMINARIES

In this section, we will introduce the global routing problem and overall flow of our router.

2.1 Global Routing Problem

As shown in Figure 1, the 3D multi-layer design is partitioned into small rectangular parts called GCells by evenly spaced horizontal and vertical grid lines. We call the edge between two GCells on the same layer wire, and the edge between two GCells with the same 2D coordinates on adjacent layers is called via. The capacity of a wire is the number of tracks that can go through this edge. There is no capacity constraints on a via, but a via utilizes routing resources on its associated wire, which will be further discussed later. Global routing problem can be defined as follows. Given a multi-layer design and a set of nets to be routed, find a path for each net to connect all its pins. Meanwhile, the wirelength and via usage should be minimized without overflow (capacity constraints violation).

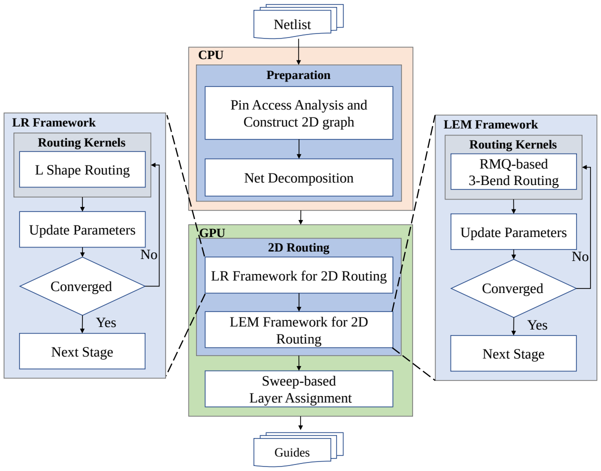


Figure 2: The overall flow of our router.

2.2 Evaluation Metrics

In this work, we adopt the same evaluation method as the ISPD 2024 GPU/ML-enhanced global routing contest [1] as follows.

Unstacked and stacked vias: As shown in Figure 1(b), unstacked vias [32] are defined as vias that establish connections with one or more horizontal or vertical wires. In contrast, stacked vias only connect to other vias. A stacked via needs to occupy half of the routing demand on its associated wires.

Quality score: Quality score is measured by the weighted sum of total wirelength, via utilization and overflow penalty,

$$Score = w_1 \cdot TotalWL + w_2 \cdot ViaCount + OverflowScore, \quad (1)$$

where $TotalWL$ is the sum of wirelength for all nets and $ViaCount$ is the total number of vias. Parameters w_1 and w_2 are input for each design to adjust the weights of different metrics. The $OverflowScore$ is the sum of $OverflowCost$ on all edges. For an edge on layer l with routing demand d and capacity c,

$$OverflowCost(c, d, l) = OvflW_l \cdot e^{s(d-c)} \quad (2)$$

, where s is 0.5 for $c > 0$, 1.5 for $c = 0$. $OvflW_l$ denotes the overflow weight for GCell edges at layer l, which is provided as input for each design. Larger $OvflW_l$ indicates a larger penalty on overflow at this layer.

3 ALGORITHM FRAMEWORK

In this section, we will first introduce our overall flow. Then, we explain the details about our LEM for routing, GPU-accelerated routing kernels, and GPU-accelerated layer assignment.

3.1 Overall Flow

Fig 2 presents the overall flow of our router, which can be divided into 3 phases: preparation, 2D routing, and layer assignment. Only the first phase runs on a CPU, and the other phases run on a GPU. After simple pin access analysis and net decomposition using FLUTE [33], we adopt the 2D soft capacity method in [13] to compact 3D graph to 2D graph. The soft capacity technique helps to get a 3D-routability-driven 2D routing result. Then, we perform 2D routing based on a Lagrangian relaxation framework and a LEM framework with different routing kernels. Finally, we perform sweep-based layer assignment to generate 3D routing guides.

3.2 Lagrangian Relaxation for Routing

Let N be the set of nets, E be the set of edges, $x_{n,e} \in \{0,1\}$ be whether net n uses edge e, Cap_e be the capacity constraints of edge e and w_e be the wirelength cost of edge e. The problem of routing of two-pin nets is formulated as an ILP problem given as follows [30]:

$$\begin{aligned}
\min_{x_{n,e}} \quad & \sum_{e \in E} w_e \sum_{n \in N} x_{n,e}, && \text{(3a)}\\
\text{s.t.} \quad & \sum_{n \in N} x_{n,e} \leq Cap_e, \forall e \in E, && \text{(3b)}\\
& \textit{connected path constraints}, && \text{(3c)}\\
& x_{n,e} \in \{0,1\}, && \text{(3d)}
\end{aligned}$$

where the detailed formats of constraints (3c) are omitted for the brevity (refer to [30] for details), which will not affect the following derivation. Let χ be the feasible set for constraints (3c) and (3d) of the problem above, i.e.,

$$\chi = \{\mathbf{x} \in \{0,1\}^{|N||E|} | \mathbf{x} \text{ satisfies (3c)}\}. \tag{4}$$

We then define the characteristic function of set χ as,

$$I_\chi(\mathbf{x}) = \begin{cases} 0 & \mathbf{x} \in \chi, \\ +\infty & \mathbf{x} \notin \chi. \end{cases} \tag{5}$$

By introducing a Lagrangian multiplier $\mathbf{y} \in R^{|E|}$ with $\forall e \in E, y_e \geq 0$ in the constraints (3b), we write the problem (3) as follows: The Lagrangian function of problem (3) can be written as [34],

$$L(\mathbf{x},\mathbf{y}) = \sum_{e \in E} w_e \sum_{n \in N} x_{n,e} + \sum_{e \in E} y_e \left(\sum_{n \in N} x_{n,e} - Cap_e \right) + I_\chi(\mathbf{x}). \tag{6}$$

Solving the dual problem

$$\max_{\mathbf{y} \geq \mathbf{0}} \min_{\mathbf{x}} L(\mathbf{x},\mathbf{y}) \tag{7}$$

will lead to the optimal solution of the problem (3). We follow the same procedure as [30] to solve (7). The basic idea is to iterate between two phases: 1) routing all nets using a routing kernel; 2) updating the Lagrangian multiplier $\mathbf{y}$ using the gradient ascent method:

$$y_e^{k+1} = \max \left\{ 0, y_e^k + \frac{1}{100k} \left(\sum_{n \in N} x_{n,e}^k - Cap_e \right) \right\}, \tag{8}$$

where k denotes the iteration. We perform L shape routing under the Lagrangian relaxation routing scheme for 8 iterations. Due to the page limit, we omit the details.

Previous studies like [29, 30, 35] leverage such Lagrangian relaxation to solve routing problems. The benefit of such a method is that all nets can be routed in parallel using the edge costs (determined by the Lagrangian multiplier) from the previous iteration, which can be extremely fast. However, our experiments show that it may oscillate and be unable to quickly reduce routing overflow, as the linear penalty in (6) does not pay enough attention to high-overflow regions. Therefore, we only use this method for L-shape routing to perform initial planning.

3.3 Linearized Exponential Multiplier Method

In this section, we propose a linearized exponential multiplier method to speedup the reduction of overflow. The basic idea is to introduce an exponential penalty function [36] for overflow to the objective of the problem (3). Then, we solve the exponentially penalized problem with a proximal linearized block coordinate descent method, the subproblems of which can be solved by routing kernels. By iteratively updating the multipliers, we can force the reduction of overflow and achieve convergence.

As the overflow cost in Equation (2) penalizes the overflow exponentially, we define an exponential penalty function $\psi : R \to R$ to align with the format of the overflow cost,

$$\psi_\rho(t) = \frac{1}{\rho}\left(e^{\rho t} - 1\right). \tag{9}$$

Then, the capacity constraint (3b) can be equivalently written as,

$$\psi_\rho \left(\sum_{n \in N} x_{n,e} - Cap_e \right) \leq 0. \tag{10}$$

We associate a multiplier $\mathbf{y} \in R^{|E|}$ with $\forall e \in E, y_e > 0$ with the capacity constraint (10) for edge e. The Lagrangian function of problem (3) can be written as

$$\begin{aligned}
& L_\rho(\mathbf{x},\mathbf{y}) \\
&= \underbrace{\sum_{e \in E} w_e \sum_{n \in N} x_{n,e} + \sum_{e \in E} y_e \psi_\rho \left(\sum_{n \in N} x_{n,e} - Cap_e \right)}_{\Phi_\rho(\mathbf{x},\mathbf{y})} + I_\chi(\mathbf{x}), && \text{(11)}
\end{aligned}$$

where $\Phi_\rho(\mathbf{x},\mathbf{y})$ is introduced for brevity of equations later. We use $x_{n,e}^k, y_e^k, \rho^k$ to denote the value of variable $x_{n,e}, y_e, \rho$ at the k-th iteration. We compute $\mathbf{x^{k+1}}$ by finding the best $\mathbf{x}$ given $\mathbf{y^k}$,

$$\mathbf{x^{k+1}} = \arg\min_{\mathbf{x}} L_{\rho^k}\left(\mathbf{x},\mathbf{y^k}\right), \tag{12}$$

and update the multipliers $\mathbf{y^{k+1}}$ as follows [36],

$$y_e^{k+1} = y_e^k \nabla_{\mathbf{x^{k+1}}} \psi_{\rho^k} \left(\sum_{n \in N} x_{n,e}^{k+1} - Cap_e \right). \tag{13}$$

Intuitively, by increasing the value of ρ, the penalty for violating the constraints increases exponentially. Meanwhile, we update the multipliers $\mathbf{y}$ to ensure that the penalty for constraints without any violations will become closer to 0. Such a method can be proved to be equivalent to the Lagrangian relaxation method in Equation (6) according to [37], but can achieve much faster convergence in satisfying the constraints.

By observing variables of different nets are only coupled in the penalty function ψ and the optimal solution of a net always uses edges around its pins, we divide N into B subsets $S_1, S_2, ..., S_B$ satisfying the following rules,

$$\begin{aligned}
S_i &= \left\{net_{i,1}, net_{i,2}, ..., net_{i,|S_i|}\right\}, && \text{(14a)}\\
S_1 \cup S_2 ... \cup S_B &= N, && \text{(14b)}\\
S_i \cap S_j &= \emptyset, \forall i \neq j, && \text{(14c)}
\end{aligned}$$

where $n_{i,j}$ denote the j-th element of S_i. Let $\mathbf{X_i}$ be the i-th block variable of $\mathbf{x}$ corresponding to S_i in (14), i.e, $\mathbf{x} = (\mathbf{X_1}; ...; \mathbf{X_B})$, and $\mathbf{X_i} = (\mathbf{x_{n_{i,1}}}; ...; \mathbf{x_{n_{i,|S_i|}}})$. For example, if the net set $N = \{n_1, n_2, n_3\}$, edge set $E = \{e_1, e_2\}$, and $S_1 = \{n_2\}, S_2 = \{n_1, n_3\}$, then we have variable set $\mathbf{x}$ and block variables $\mathbf{X_1}, \mathbf{X_2}$,

$$\begin{aligned}
\mathbf{x} &= \left\{x_{n_1,e_1}, x_{n_1,e_2}, x_{n_2,e_1}, x_{n_2,e_2}, x_{n_3,e_1}, x_{n_3,e_2}\right\}, \\
\mathbf{X_1} &= \left\{x_{n_2,e_1}, x_{n_2,e_2}\right\}, && \text{(15)}\\
\mathbf{X_2} &= \left\{x_{n_1,e_1}, x_{n_1,e_2}, x_{n_3,e_1}, x_{n_3,e_2}\right\}.
\end{aligned}$$

In other words, $\mathbf{X_i}$ is a subset of variables corresponding to a subset of nets.

Let χ_i be the feasible set of $\mathbf{X_i}$. We solve the problem (12) by a Block Coordinate Descent (BCD) algorithm, where we minimize L_{ρ^k} in order of $\mathbf{X_1}, \mathbf{X_2}, ..., \mathbf{X_B}$, by solving each subproblem,

$$\begin{aligned} &\arg\min_{\mathrm{X_i}} L_{\rho^k}\left(\mathrm{X_1^{k+1}}, ..., \mathrm{X_{i-1}^{k+1}}, \mathrm{X_i}, \mathrm{X_{i+1}^{k}}, ..., \mathrm{X_B^k}, \mathrm{y^k}\right) \\ =&\arg\min_{\mathrm{X_i}} \Phi_{\rho^k}\left(\mathrm{X_1^{k+1}}, ..., \mathrm{X_{i-1}^{k+1}}, \mathrm{X_i}, \mathrm{X_{i+1}^{k}}, ..., \mathrm{X_B^k}, \mathrm{y^k}\right) + I_{\chi_i}\left(\mathrm{X_i}\right). \end{aligned} \tag{16}$$

As Φ is differentiable and the projection operator of χ_i is easy to calculate, we adopt the proximal linear method [38] as a linearization technique to solve the subproblem (16), i.e.

$$\mathbf{X_i^{k+1}} \in \arg\min_{\mathrm{X_i} \in \chi_i}\left(\left\langle \mathbf{X_i} - \mathbf{X_i^k}, \nabla_{\mathrm{X_i}}\Phi_{\rho^k}\right\rangle + \frac{1}{2\tau}\|\mathbf{X_i} - \mathbf{X_i^k}\|_2^2\right), \tag{17}$$

where $\tau > 0$ is the step size. Note that I_{χ_i} is moved to constrain the definition domain of $\mathbf{X_i}$, i.e., $\mathbf{X_i} \in \chi_i$. The intuition of the proximal linear method is to find the next solution $\mathbf{X_i^{k+1}}$ near the neighborhood of the current solution $\mathbf{X_i^k}$ and meanwhile minimize the objective. We use the symbol '$\in$' instead of '$=$' in (17) because there might be multiple optimal solutions for the subproblem. If one defines the projection operator by,

$$P_\chi(\mathbf{v}) = \arg\min_{\mathbf{u}\in\chi} \|\mathbf{u} - \mathbf{v}\|_2^2, \tag{18}$$

then we obtain the following equivalent form of (17),

$$\mathbf{X_i^{k+1}} \in P_{\chi_i}\left(\mathbf{X_i^k} - \tau\nabla_{\mathrm{X_i}}\Phi_{\rho^k}\right). \tag{19}$$

Note that

$$P_\chi(\mathbf{v}) = \arg\min_{\mathbf{u}\in\chi} \|\mathbf{u}\|_2^2 - 2\mathbf{v}^T\mathbf{u} + \|\mathbf{v}\|_2^2, \tag{20a}$$

$$= \arg\min_{\mathbf{u}\in\chi} \mathbf{1}^T\mathbf{u} - 2\mathbf{v}^T\mathbf{u}, \text{as } u_i \in \{0,1\} \text{ when } \mathbf{u} \in \chi, \tag{20b}$$

$$= \arg\min_{\mathbf{u}\in\chi} (\mathbf{1} - 2\mathbf{v})^T\mathbf{u}. \tag{20c}$$

We can rewrite the subproblem in (19) as following,

$$\mathbf{X_i^{k+1}} \in \arg\min_{\mathrm{X_i}\in\chi_i}\left(1 - 2\left(\mathbf{X_i^k} - \tau\nabla_{\mathrm{X_i}}\Phi_{\rho^k}\right)\right)^T\mathbf{X_i}. \tag{21}$$

Note that the shortest path problem for net n can be formulated as follows,

$$\min \quad \sum_{e\in E} cost_{n,e}x_{n,e}, \tag{22a}$$

$$\text{s.t.} \quad \textit{connected path constraints}, \tag{22b}$$

$$x_{n,e} \in \{0,1\}, \tag{22c}$$

where $cost_{n,e}$ is edge e's cost for net n. Then, by recalling the definition of χ in (4), solving the problem (21) is equivalent to solving a set of shortest path problems for the subset of nets S_i with the edge costs in the k-th iteration for each net n defined as,

$$\begin{aligned} cost_{n,e} &= 1 - 2\left(x_{n,e}^k - \tau\nabla_{x_{n,e}}\Phi_{\rho^k}\right), \\ &= 1 - 2x_{n,e}^k + 2\tau\left(w_e + y_e^k\nabla_{x_{n,e}}\psi_{\rho^k}\right), \end{aligned} \tag{23}$$

which can be solved by well-designed routing kernels.

The overall exponential multiplier procedure for the routing problem is summarized in Algorithm 1. Suppose that we have a `RoutingKernel` that can finish the routing for a subset (a.k.a batch) of nets given the edge costs. In each iteration, we apply the `RoutingKernel` for nets in order of $S_1, S_2, ..., S_B$ (line 3). Then, we update multiplier $\mathbf{y}$ in line 6 and penalty factor ρ in line 7. In the experiments, we set $k_{max} = 3, p_{max} = 1$ for efficiency. We set $y_e^0 = \frac{1}{100L}\sum_{l=1}^{L} OvflW_l, \rho^0 = 0.05, \sigma = 2, \tau = 100$, where L is the number of layers and $OvflW_l$ is a given parameter in the input file, as described in Section 2.2. Ideally, `RoutingKernel` needs to solve subproblem (16) optimally. In practice, we can balance runtime and optimality with various fast approximation algorithms, as introduced in the next section.

Algorithm 1: LEM method for routing

```
   Input: $\mathbf{x^0}, \mathbf{y^0}, \rho^0, \sigma, \tau, \mathbf{w}, \mathbf{Cap}, k_{max}, p_{max}$
   Output: $\mathbf{x^{k_{max}}}$
1  for $k \leftarrow 0$ to $k_{max} - 1$ do
2      for $p \leftarrow 0$ to $p_{max} - 1$ do
3          for $i \leftarrow 1$ to $B$ do
4              // Solve problem (16) with edge costs (23)
               $\mathbf{X_i^{k+1}} \leftarrow$ RoutingKernel($\mathbf{X_1^{k+1}}, ..., \mathbf{X_{i-1}^{k+1}},$
                         $\mathbf{X_i}, \mathbf{X_{i+1}^k}, ..., \mathbf{X_B^k}, \mathbf{y^k}, \rho^k, \tau, \mathbf{w}, \mathbf{Cap}$);
5      for $e \in E$ do
6          Update $y_e^{k+1}$ according to (13);
7      $\rho^{k+1} \leftarrow \sigma\rho^k$;
```

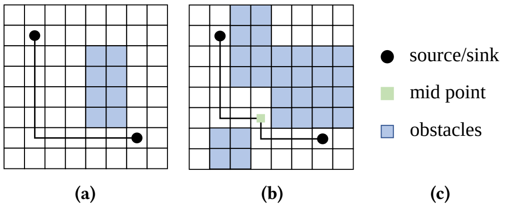


Figure 3: (a) L shape routing. (b) 3-bend routing.

3.4 GPU-Accelerated Routing Kernels

Pattern routing is widely used in modern global routers [12, 13, 15, 18, 19, 24, 25]. It restricts the routing solution into some specific shapes to reduce runtime. In this work, we adopt L shape and 3-bend patterns to achieve high-quality routing solutions, as shown in Figure 3.

3.4.1 L shape routing. L shape is the most widely pattern used in previous routers. As shown in Figure 3(a) , within a $M \times N$ bounding box, L shape routing has at most 2 candidate paths, so it is not quite time-consuming to perform L shape routing. We assign one thread on GPU to do L pattern routing for each net.

3.4.2 3-bend routing. 3-bend routing is used in many 2D routers, such as FastRoute 4.0 [10] and SPRoute 2.0 [13]. It can provide better routing solution than L shape pattern routing. We will first introduce how 3-bend routing was implemented in the prior work, and then propose a GPU-friendly version of 3-bend routing algorithm for solving a batch of nets.

In Figure 4, we show an example of how to use the 3-bend pattern to route nets. Figure 4(a) draws two nets for routing and their bounding boxes. A typical 3-bend routing algorithm needs to enumerate mid points to find the minimum cost to connect to the source and sink using L shape patterns, respectively. Figure 4(b) annotates the mid points for nets n_1 and n_2, numbered as $\{0, 1, ..., 5\}$ and $\{6, 7, ..., 20\}$, respectively. Figure 4(c) calculates the minimum costs for all mid points and find one with the smallest cost for each net. Finally, we connect the source and sink with the best path in Figure 4(d). The most time-consuming part is calculating the minimum cost and selecting the best one. First, we need to calculate the prefix sum of the horizontal (vertical) edge cost for each net. With those prefix sums, we can get the

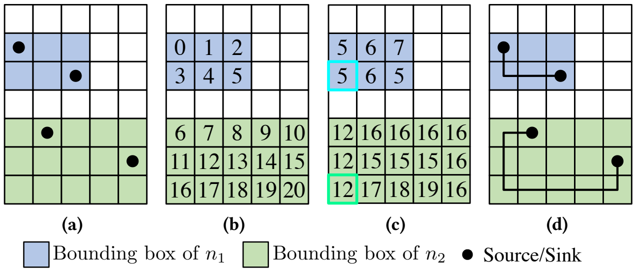


Figure 4: Example for routing 2 nets with 3-bend patterns. (a) Two nets and their bounding boxes; (b) number all candidate mid points; (c) calculate costs for all mid points and find the one with the smallest cost for each net; (d) route nets with the best paths.

cost of the horizontal (vertical) path in $O(1)$, so we can calculate the cost of a 3-bend path in $O(1)$. After getting all the cost of each mid point, we select the smallest cost sequentially. For a $M \times N$ bounding box, there are $M \times N$ mid points, so the time complexity of the 3-bend routing algorithm on CPU in previous works is $O(MN)$ [10, 13].

The procedure of the above algorithm can be summarized by solving the following two subproblems:

(1) Calculating costs of mid points by prefix sum of edge costs.
(2) Finding the mid point with minimum cost for each net.

In a large-scale global routing problem, we need to perform 3-bend routing for a huge number of nets. A simple strategy like net-level parallelization is inefficient on GPU due to imbalanced workloads from heterogeneous net bounding boxes. Hence, new algorithms are desired to speedup the computation on GPU.

Let N be the set of nets, (lx_n, ly_n, hx_n, hy_n) be the bounding box for net n, and we define l_n, r_n satisfies,

$$N = \left\{n_1, n_2, ..., n_{|N|}\right\}, \tag{24a}$$

$$r_{n_i} - l_{n_i} = \left(hy_{n_i} - ly_{n_i} + 1\right) \times \left(hx_{n_i} - lx_{n_i} + 1\right), \forall n_i \in N, \tag{24b}$$

$$l_{n_1} = 0, l_{n_i} = r_{n_{i-1}}, \forall i = 2, ..., |N|. \tag{24c}$$

The above equation essentially tries to flatten the mid points within the bounding box for each net into a range of an array. Given the example in Figure 4, we have $l_{n_1} = 0, r_{n_1} = 6$ and $l_{n_2} = 6, r_{n_2} = 21$. Let $A[i]$ be the cost of the candidate path of the i-th mid point. Then, $A[i, j)$ denotes the costs of mid points from i to $j-1$. For example, $A[0, 6)$ is the costs of mid points $\{0, 1, \ldots, 5\}$. We show in Figure 5 and Algorithm 2 on how the flattened array helps to handle two subproblems mentioned above with three steps.

For the first subproblem, the bottleneck lies in calculating A, which is equivalent to computing the prefix sum within the bounding box of each net. The workload for solving this subproblem can be very imbalanced if we perform net-level parallelization, as the bounding box of each net varies significantly. With the flattened formulation, we can merge the horizontal (vertical) prefix sum problems for a batch of nets into a *big* prefix sum problem in an array with $r_{n_{|N|}}$ elements, and meanwhile, still maintain the mid points with minimum costs for the second subproblem. We can get the prefix sum with GPU prefix scan and thus calculate A with time complexity $O\left(log_2 r_{n_{|N|}}\right)$ without any workload imbalance issues, shown in Step 1 in Figure 5 and Algorithm 2.

For the second subproblem, with the flattened array A, we can formulate it as a Range Minimum Query (RMQ) problem. That is, the minimum cost of net n is the minimum element of the interval $A[l_n, r_n)$. For instance, in Figure 4, the best solutions for net n_1 and net n_2 are the minimum elements of $A[0, 6)$ and $A[6, 21)$, respectively. Inspired by a classical data structure called segment tree for solving the RMQ problem, we propose an efficient algorithm to find the minimum element of $A[l_n, r_n)$, corresponding to Step 2 and 3 in Figure 5 and Algorithm 2. We define an interval of array $A\left[i, i+2^j\right)$ as a segment, where $i \bmod 2^j = 0$. Let $d(i, j)$ be the minimum of segment $A\left[i, i+2^j\right)$. Step 2 of our algorithm is to calculate d, and Step 3 is to find the minimum cost for each net with d.

Function `calcMinOfSegments` in line 4 of Algorithm 2 summarizes how to get array d on GPU with a divide and conquer algorithm for Step 2. We first initialize the smallest segment $d(i, 0)$ to $A(i)$. Then, for every level j, segment $d(i, j)$ depends on the last level $j-1$, $d(i, j-1)$ and $d\left(i + 2^{j-1}, j-1\right)$. We can calculate $d(i, j)$ at the same level j concurrently on GPU, so Step 2 runs in time $O\left(log_2 r_{n_{|N|}}\right)$. Considering the example in Figure 5, we calculate $\{d(0,1), \ldots, d(18,1)\}, \{d(0,2), \ldots, d(16,2)\}, \ldots, \{d(0,4)\}$ concurrently in 4 iterations. `calcMinCostOfNets` in line 13 of Algorithm 2 introduces how to use d to get the minimum of $A[l, r)$ for Step 3. We allocate one thread for each net. Take net n_1 in Figure 5 as an example. It checks segments $\{[0,1), [0,2), [0,4), [4,8), [4,6)\}$ and merge the minimum of $\{[0,4), [4,6)\}$ (i.e., $\min\{d(0,2), d(4,1)\}$) to get the minimum of $A[0, 6)$.

Lemma 3.1. *Lines 13-28 in Function* `calcMinCostOfNets` *can finish in $O(log_2(R-L))$ steps.*

Proof. For problem $\min A[L, R)$, we always try to find the longest segment $\left[L, L+2^j\right) \subseteq [L, R)$ and merge $d(L, j)$ to the result. In the next iteration, we try to solve the subproblem $\min A[L + 2^j, R)$. Let k_1, k_2 satisfy $L \bmod 2^{k_1} = 0$, $L \bmod 2^{k_1+1} > 0$, $2^{k_2} \leq (R-L) < 2^{k_2+1}$. We define $k_1 = \lfloor log_2(R-L) \rfloor$ when $L = 0$. In other words, k_1 denotes the longest segment corresponding to L, and k_2 denotes the largest length does not exceed the boundary R. We have $j = \min\{k_1, k_2\}$ and $k_1, k_2 \leq \lfloor log_2(R-L) \rfloor$.

There are two possible cases in every iteration and these two cases correspond to the first and second parts of our algorithm :

(1) $k_1 \leq k_2$: In the next iteration, $\left(L + 2^{k_1}\right) \bmod 2^{k_1+1} = 0$, we have $k_1' \geq k_1 + 1$. So after at most $O(log_2(R-L))$ iterations, we have $k_1 > k_2$.
(2) $k_1 > k_2$: In the next iteration, $\left(L + 2^{k_2}\right) \bmod 2^{k_2} = 0, R - L - 2^{k_2} < 2^{k_2}$, we have $k_1' \geq k_2$ and $k_2' < k_2$, so $k_1' > k_2'$. That means, if we have $k_1 > k_2$ in one iteration, we will have $k_1 > k_2$ in later iterations. The subproblem $\left[L + 2^j, R\right)$ has the length less than $\frac{1}{2}(R-L)$, so after at most $O(log_2(R-L))$ iterations, we can solve this problem.

Based on the above analysis, we know `calcMinCostOfNets` can finish in $O(log_2(R-L))$ steps. □

Combining the runtime complexity of the three steps in Algorithm 2, we have the following theorem.

Theorem 3.2. *Algorithm 2 has $O\left(log_2 r_{n_{|N|}}\right)$ time complexity.*

As our `RMQ-based` 3-bend routing algorithm runs in logarithmic complexity for each net on GPU, it significantly mitigates the workload imbalance issue, compared with the prior algorithm [10, 13] running in linear complexity.

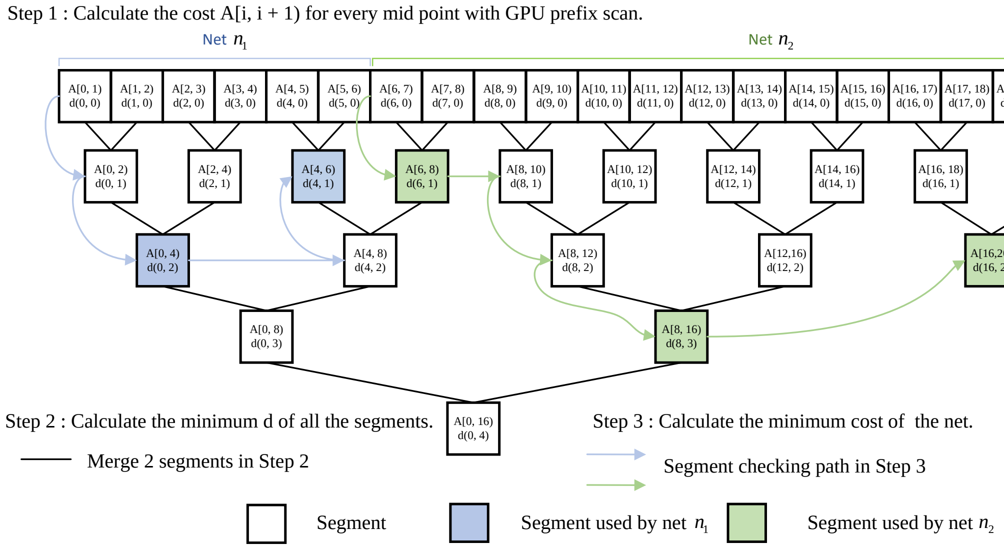


Figure 5: Our `RMQ-based` 3-bend routing algorithm. Step 1 : calculate the cost $A\,[i, i+1)$ for every mid point. Step 2 : calculate the minimum d of all the segments. Step 3 : calculate the minimum cost of each net. Take net n_1 as an example. At the beginning, $A\,[0,1)$ is checked. Since $A\,[0,2)$ is the subset of $A\,[0,6)$, we move to $A\,[0,2)$, and then move to $A\,[0,4)$. The longer `segment` $A\,[0,8)$ exceeds the boundary 6, so we do not move to it. We merge $d\,(0,2)$ into the result, and then move to $A\,[4,8)$. We move to $A\,[4,6)$ and it is the subset of $A\,[0,6)$. We merge $d\,(4,1)$ into the result. The solution of net n_1 is $\min\{d\,(0,2)\,, d\,(4,1)\}$.

Algorithm 2: `RMQ-based` 3-Bend Routing on GPU

```
Input: a batch of nets $N = \{n_1, n_2, \ldots, n_{|N|}\}$
Output: 2D routing solutions $res_1, res_2, \ldots, res_{|N|}$
Step 1 : flatten mid points within bounding boxes of nets in
    $N$ to get $l, r$, and calculate costs $A$ with GPU prefix scan;
Step 2 : $d \leftarrow$ calcMinOfSegments$\left(A, r_{n_{|N|}}\right)$;
Step 3 : $res \leftarrow$ calcMinCostOfNets$(d, l, r)$;
Function calcMinOfSegments$(A, L)$:
    $i \leftarrow$threadIdx;
    $d\,(i,0) \leftarrow A\,(i)$;
    for $j \leftarrow 1$ to $log_2 L$ do
        if $i \bmod 2^j = 0$ and $i + 2^{j-1} \le L$ then
            $d\,(i,j) \leftarrow \min\left\{d\,(i, j-1)\,, d\left(i + 2^{j-1}, j-1\right)\right\}$;
        sync all threads;
        $j \leftarrow j + 1$;
    return $d$ after all threads finish;
Function calcMinCostOfNets$(d, l, r)$:
    $i \leftarrow$ threadIdx $+ 1$;
    $L \leftarrow l_{n_i}, R \leftarrow r_{n_i}, res_i \leftarrow +\infty$;
    while $L < R$ do                                // First part
        while $L \bmod 2^{j+1} = 0$ and $L + 2^{j+1} \le R$ do
            $j \leftarrow j + 1$;
        if $L + 2^j > R$ then
            break;
        $res_i \leftarrow \min\{res_i, d\,(L, j)\}$;
        $L \leftarrow L + 2^j$;
    while $L < R$ do                                // Second part
        while $L \bmod 2^j > 0$ or $L + 2^j > R$ do
            $j \leftarrow j - 1$;
        $res_i \leftarrow \min\{res_i, d\,(L, j)\}$;
        $L \leftarrow L + 2^j$;
    return $res$ after all threads finish;
```

3.5 GPU-Accelerated Layer Assignment

After 2D routing, we need to generate 3D routing guides. Layer assignment tries to decide which layer a 2D wire should be placed and the positions of vias used to connect wires in different layers. Meanwhile, number of vias and overflow penalty should be considered. We first introduce a typical layer assignment algorithm in previous work [13], and then propose our GPU-friendly version of layer assignment.

Let G_i be the i-th GCell in the 2D path, and $G_{i,j}$ be the GCell on j-th layer with the same 2D position as G_i. For example, suppose that there is a pin in GCell $G_{1,1}$ and $G_{1,2}$ and a pin to connect in $G_{5,2}$. Figure 6(a) is the 2D routing solution from G_1 to G_5. The objective of layer assignment is to find a 3D path from source to sink based on the 2D routing solution while minimizing via usage and overflow penalty. Figure 6(b) shows a possible 3D path from $\left\{G_{1,1}, G_{1,2}\right\}$ to $\left\{G_{5,2}\right\}$. Our layer assignment algorithm is based on a directed grid graph model in Figure 6(c). The directed grid graph transforms the 3D problem into a 2D problem. One dimension is arranged in the directional order along the 2D path and the other dimension is arranged according to the layers. All the possible 3D paths can be found in the directed grid graph.

For an edge in the grid graph, we use a cost function considering wire / via usage and overflow cost. There are two types of edges in the grid graph and their cost are:

(1) $G_{i,j} \longrightarrow G_{i+1,j}$: a wire in one layer.

$$cost_e = \begin{cases} +\infty, e \text{ is a non-preferred edge,} \\ l_e + penalty\,(c_e, d_e, l_e)\,, \text{others,} \end{cases} \tag{25}$$

where l_e is the wirelength cost of edge e, and $penalty\,(c_e, d_e, l_e)$ is $OverflowCost\,(c_e, d_e + 1, l_e) - OverflowCost\,(c_e, d_e, l_e)$. $OverflowCost\,(c, d, l)$ is defined in (2).

(2) $G_{i,j} \longleftrightarrow G_{i,j+1}$: a via connecting two neighboring layers. $cost_e$ is defined as the sum of unit via cost and the change in overflow cost if the via is used.

We select the shortest path from $\{G_{1,1}, G_{1,2}\}$ to $\{G_{5,2}\}$ as the 3D routing solution of 2D path $G_1 \longrightarrow G5$. Note that the problem

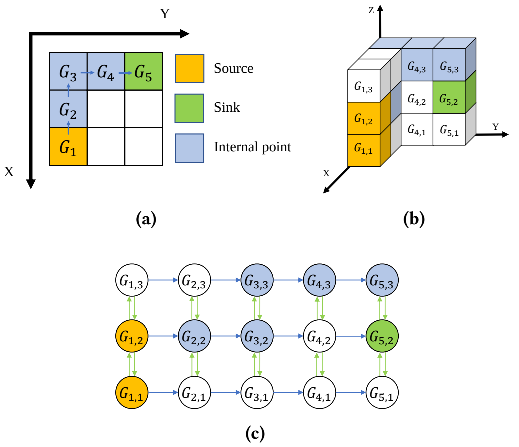


Figure 6: Example for layer assignment. (a) 2D routed path from source to sink. (b) The 3D path from source to sink after layer assignment. (c) Directed grid graph constructed to solve the layer assignment problem.

is based on a “rectangular" grid graph, we can use a dynamic programming-based algorithm instead of a general shortest path algorithm to solve it. The dynamic programming-based algorithm runs in time $O(KL)$, where K is the 2D path length and L is the number of layers. But due to the large scale of those routing cases, we need a faster method on GPU.

Algorithm 3: `Sweep`-based Layer Assignment on GPU

```
Input: 2D Path length $K$, $G_1, ..., G_K$, pin location
       $botLayer_1, topLayer_1, botLayer_K, topLayer_K, f,$
       $prev, cost$, Layer Number $L$, Threshold $\theta$
Output: 3D path with minimum cost
for $i \leftarrow 1$ to $K$ do
    for $j \leftarrow 1$ to $L$ do
        $f_{i,j} \leftarrow +\infty$;
for $j \leftarrow botLayer_1$ to $topLayer_1$ do
    $f_{1,j} \leftarrow 0$;
for $counter \leftarrow 1$ to $\theta$ do
    $f, prev \leftarrow$ VerticalSweep($f, prev, cost, K, L$);
    $f, prev \leftarrow$ HorizontalSweep($f, prev, cost, K, L$);
for $j \leftarrow botLayer_K$ to $topLayer_K$ do
    Update the best solution;
Get 3D path according to the best solution and $prev$;
```

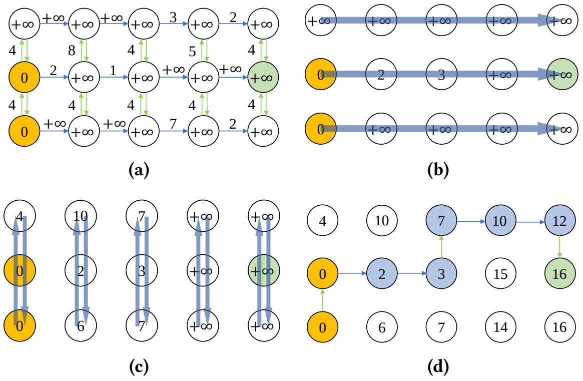


Figure 7: (a) Source, sink, and edge costs in the directed grid graph. (b) Horizontal sweep. (c) Vertical sweep. (d) Final solution after several sweeps.

It is hard to accelerate the dynamic programming-based algorithm with GPU because of the heavy data dependency. So we need a new algorithm to deal with this shortest path algorithm. We are inspired by the `sweep-based` shortest path algorithm on 3D grid graph proposed by [22]. A `sweep` is to update the shortest distances to all G-cells with edges in one direction. Horizontal (vertical) sweeps only use horizontal (vertical) edges. After doing the `sweep` operation horizontally and vertically alternatively for several iterations, the shortest distance to every GCell can be found correctly. `Sweep` can be implemented by GPU prefix scan algorithm. Figure 7 illustrates the process of our GPU layer assignment algorithm. After preparing edge costs and initial vertex distance in Figure 7(a), we do horizontal and vertical `sweep` alternatively for several iterations in Figure 7(b) and Figure 7(c). Finally we get the shortest distance of every vertex in Figure 7(d).

4 EXPERIMENTAL RESULTS

We implement our router in `C++` and `CUDA`. We enable CPU multi-threading with `Taskflow` [39] and use `NVIDIA CUB` to compute prefix scan on GPU. All the experiments are conducted on a Linux server with 32-core Intel Xeon Gold CPU@2.90GHz and one NVIDIA A100 GPU. The benchmarks are all from ISPD 2024 GPU/ML-enhanced global routing contest [1] with up to around 50 million cells and 60 million nets. We obtain the binaries from the top-3 contest winners and conduct experiments on our machine for comparison. The official evaluator from the ISPD 2024 contest is used to evaluate the solutions and report scores.

We compare with the contest winners in Table 1. Our router can achieve about 4.8%-5.8% better quality scores and $1.62\times - 2.07\times$ speed-up on average. Especially in large cases with almost 60 million nets, our router can achieve more than 4% better quality score compared with the best contest solution, and finish routing in around 10 minutes. The breakdown of quality scores is listed in Table 2. It shows that our router has the least overflow among all routers, i.e., 15.8%, 25.5%, and 28.0% less overflow than the contest winners, respectively. Our wirelength is comparable to the contest winners as well. Although our via numbers are slightly higher than the 2nd and 3rd winners, our overflow is significantly less than their solutions. We ascribe such benefits to the LEM method that can effectively reduce overflow with a global view.

In Figure 8, we compare the runtime between four routers scaling with the number of nets. Our router achieves the best scalability with the problem size growing. Especially in large cases, we can achieve 1.58×, 1.87×, and 2.10× faster than the contest winners, respectively. We ascribe the much shorter runtime of our router to the efficient GPU-accelerated routing kernels that can explore a huge number of possible paths in a short time.

We also compare the normalized total and maximum overflow at the 2D routing stage in Figure 9(a)-9(b), respectively. `LR` denotes the Lagrangian relaxation model in previous studies [29, 30, 35]. `LR+LEM` denotes the two-stage Lagrangian relaxation and LEM model we adopt in Section 3.3. We can see that `LR+LEM` leads to a more stable convergence with much less overflow. We observe that in each iteration, the `LR` method solves the routing subproblem without awareness of the changes in routing resources, which may cause slow convergence and oscillation when switching to a new pattern. Figure 9(c) shows the runtime breakdown of our router on case `terapool_cluster_h`. We can see 2D routing and layer assignment take around 67% of the overall runtime. I/O

Table 1: Comparison of quality score and runtime with top-3 winners in the ISPD 2024 contest.

Design[†]	#Nets	Quality Score				Runtime[‡] (s)			
		1st	2nd	3rd	Ours	1st	2nd	3rd	Ours
ariane_51_p	126873	22536394	22697143	22501453	**22484547**	2.6	2.6	**1.3**	2.5
ariane_68_p	127026	19789143	20099496	19897356	**19734912**	2.5	2.0	**1.2**	2.4
ariane_68_h	128282	22602876	23093501	22821289	**22561640**	2.9	2.0	**1.4**	2.5
mempool_tile_p	135919	15241650	15432389	15280353	**15195056**	2.6	2.1	**1.3**	2.8
mempool_tile_h	135666	13827142	14133269	13867124	**13773881**	2.3	1.9	**1.1**	2.6
nvdla_p	176755	48257027	48837685	48257010	**48108771**	4.3	**3.2**	3.4	3.5
nvdla_h	174731	43195979	44141552	43295092	**43031318**	3.9	**2.9**	3.6	3.1
blackparrot_p	768239	113562198	113321049	**112592863**	112985592	13.8	15.9	14.5	**12.4**
blackparrot_h	824756	113109620	110986781	111778586	**110140593**	14.2	8.8	**7.2**	12.5
mempool_group_p	3252596	398169317	411758419	403915333	**396514188**	37.0	25.3	20.5	**18.4**
mempool_group_h	3200973	383637652	395488859	388200338	**381952598**	36.0	24.0	20.4	**18.5**
mempool_cluster_p	10612686	1626227314	1665748885	1639553927	**1616064157**	157.8	90.0	166.9	**74.3**
mempool_cluster_h	10612686	1782191834	1824527054	1795524941	**1775052604**	159.4	97.4	184.8	**74.2**
terapool_cluster_p	59271897	19609525592	19684091476	19730043753	**18468687634**	1590.6	1944.9	2129.2	**1080.7**
terapool_cluster_h	59271897	12521927637	12676067016	12597498026	**11995425076**	1085.4	1205.1	1416.3	**607.7**
Average		2448920092	2471361638	2464335163	**2336114171**	207.7	228.5	264.9	**127.9**
Ratio		1.048	1.058	1.055	**1.000**	1.624	1.787	2.071	**1.000**

[†] ⋆_p and ⋆_h denote public and hidden cases, respectively.

[‡] We observe that runtime varies on different machines. For example, on terapool_cluster_h, the ISPD 2024 contest reported that the 2nd team is the fastest [1], while on our machine, the 1st team is slightly faster than the 2nd team. Such variation does not affect the conclusion of this paper, as our router runs much faster than all three teams on large cases with millions of nets.

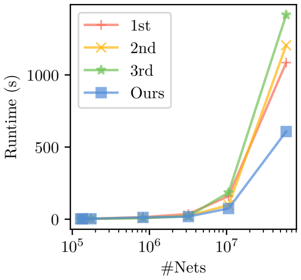
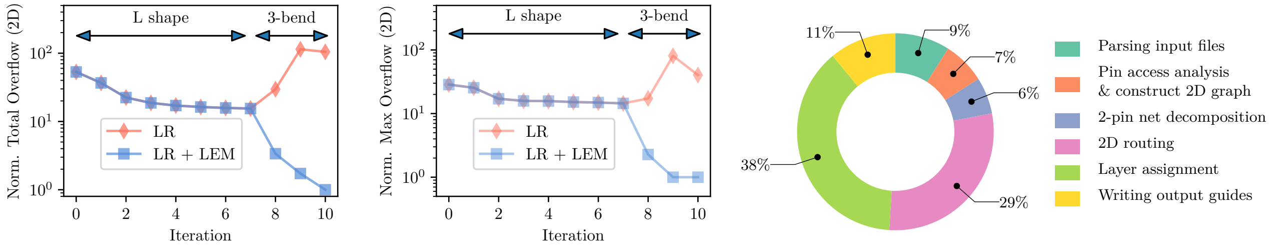


Figure 8: Runtime scaling with the number of nets (only hidden (⋆_h) cases are plotted for clear visualization).

Table 2: Comparison of quality score breakdown with top-3 winners in the ISPD 2024 contest.

Design	Wirelength				Via				Overflow			
	1st	2nd	3rd	Ours	1st	2nd	3rd	Ours	1st	2nd	3rd	Ours
ariane_51_p	**9231428**	9282664	9265081	9233921	3030032	2998676	**2924628**	2989296	10274934	10415803	10311744	**10261330**
ariane_68_p	**9325588**	9441111	9418417	9333438	2883136	2925264	**2783144**	2844740	7580419	7733121	7695795	**7556734**
ariane_68_h	**11931753**	12051687	12021241	11937576	2906688	2959392	**2813744**	2875672	7764435	8082423	7986304	**7748393**
mempool_tile_p	**8336290**	8390542	8374632	8338598	3393092	3347480	**3287436**	3341728	**3512267**	3694367	3618286	3514729
mempool_tile_h	**7541039**	7601381	7578562	7544046	3346788	3308184	**3233024**	3287124	**2939315**	3223704	3055538	2942711
nvdla_p	**21173333**	21453123	21359327	21242061	4534604	4451456	**4354664**	4464608	22549090	22933106	22543019	**22402101**
nvdla_h	**21472301**	21840287	21693203	21537535	4661336	4586500	**4455428**	4581360	17062342	17714764	17146460	**16912423**
blackparrot_p	**58016217**	58151649	58219164	58060585	19651620	19295072	**19112608**	19472196	35894361	35874328	**35261090**	35452812
blackparrot_h	**55573071**	55847570	55751773	55596948	19738252	19505444	**19182208**	19513832	37798298	35633766	36844605	**35029813**
mempool_group_p	**259242540**	262602848	261349799	259468578	74230960	75211240	**70841780**	72532976	64695817	73944331	71723754	**64512634**
mempool_group_h	**247299729**	250445950	249498393	247576859	73566828	74362960	**70274012**	71919096	62771095	70679950	68427932	**62476937**
mempool_cluster_p	**1087812280**	1093851156	1090773738	1089016864	274090104	264399732	**258033820**	266228272	264324930	307497997	290746369	**260819021**
mempool_cluster_h	**1185865082**	1190412465	1189656494	1186752708	282548572	274145704	**270441852**	277553000	313778180	359968885	335426595	**310746896**
terapool_cluster_p	**12167900631**	12292803029	12164444468	12210612912	1968847968	1533513212	**1499561196**	1642840776	5472776994	5857775235	6066038089	**4615233946**
terapool_cluster_h	**7963801629**	8039402094	7970278976	7984351151	1690296724	1488566820	**1442567228**	1529353596	2867829285	3148098101	3184651822	**2481720329**
Average	**1540968194**	1555571837	1541978885	1545373585	295181780	251571809	**244924451**	261586552	612770117	664217992	677431827	**529155387**
Ratio	**0.997**	1.007	0.998	1.000	1.128	0.961	**0.936**	1.000	1.158	1.255	1.280	**1.000**



(a)

(b)

(c)

Figure 9: (a) Comparison of normalized total overflow and (b) max overflow (evaluated at 2D routing stage) between LR and LR+LEM routing frameworks. (c) Runtime breakdown in percentage of our router on terapool_cluster_h.

takes around 20% runtime, as those files take around 31GB disk space. The preparation takes 13% runtime running on CPU only.

5 CONCLUSION

In this paper, we design a 2D global router which can generate high-quality routing solutions efficiently to handle large scale routing cases with up to 1 billion GCells and 60 million nets in ISPD 2024. A novel global routing framework based on LEM is introduced to optimize both wirelength and overflow. To efficiently explore a huge number of candidate solutions for millions of nets, we design several GPU-accelerated routing kernels. Finally, we use a sweep-based layer assignment algorithm on GPU to generate 3D routing solutions. Compared with top-3 winners in the ISPD 2024 contest, we achieved around 1.62× to 2.07× speed-up and around 4.8%-5.8% better quality score on average, and the experimental result shows that our router has a strong scalability.

ACKNOWLEDGEMENTS

This work was supported in part by the National Science Foundation of China (Grant No. T2293700, T2293701, 12331010 and 12288101), the Natural Science Foundation of Beijing, China (Grant No. Z230002), and the 111 Project (B18001).

REFERENCES

[1] R. Liang, A. Agnesina, W.-H. Liu, and H. Ren, "Gpu/ml-enhanced large scale global routing contest," in *Proceedings of the 2024 International Symposium on Physical Design*, ser. ISPD '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 269–274. [Online]. Available: https://doi.org/10.1145/3626184.3639693

[2] C. J. Alpert, D. P. Mehta, and S. S. Sapatnekar, *Handbook of Algorithms for Physical Design Automation*. CRC press, 2008.

[3] M. Pan and C. Chu, "Fastroute: A step to integrate global routing into placement," in *2006 IEEE/ACM International Conference on Computer Aided Design*, 2006, pp. 464–471.

[4] X. He, T. Huang, W.-K. Chow, J. Kuang, K.-C. Lam, W. Cai, and E. F. Y. Young, "Ripple 2.0: High quality routability-driven placement via global router integration," 2013, pp. 152:1–152:6.

[5] M.-K. Hsu, Y.-F. Chen, C.-C. Huang, S. Chou, T.-H. Lin, T.-C. Chen, and Y.-W. Chang, "NTUplace4h: A novel routability-driven placement algorithm for hierarchical mixed-size circuit designs," vol. 33, no. 12, pp. 1914–1927, 2014.

[6] T. Lin, C. Chu, J. R. Shinnerl, I. Bustany, and I. Nedelchev, "POLAR: A high performance mixed-size wirelengh-driven placer with density constraints," vol. 34, no. 3, pp. 447–459, 2015.

[7] 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.

[8] 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.

[9] L. Liu, B. Fu, S. Lin, J. Liu, E. F. Young, and M. D. Wong, "Xplace: An extremely fast and extensible placement framework," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–1, 2023.

[10] 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.

[11] M. Cho, K. Lu, K. Yuan, and D. Z. Pan, "Boxrouter 2.0: A hybrid and robust global router with layer assignment for routability," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 14, no. 2, Apr. 2009.

[12] 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.

[13] 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.

[14] J. A. Roy and I. L. Markov, "High-performance routing at the nanometer scale," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 6, pp. 1066–1077, 2008.

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

[16] 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.

[17] C. Y. Lee, "An algorithm for path connections and its applications," *IRE Transactions on Electronic Computers*, vol. EC-10, no. 3, pp. 346–365, 1961.

[18] R. Kastner, E. Bozorgzadeh, and M. Sarrafzadeh, "Pattern routing: use and theory for increasing predictability and avoiding coupling," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 21, no. 7, pp. 777–790, 2002.

[19] J. Liu and E. F. Young, "Edge: Efficient dag-based global routing engine," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*, 2023, pp. 1–6.

[20] J. Wang, J. Mai, Z. Di, and Y. Lin, "A robust fpga router with concurrent intra-clb rerouting," in *2023 28th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2023, pp. 529–534.

[21] Y. Han, K. Chakraborty, and S. Roy, "A global router on gpu architecture," in *2013 IEEE 31st International Conference on Computer Design (ICCD)*, 2013, pp. 78–84.

[22] 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.

[23] X. Jiang, J. Wang, Y. Lin, and Z. Wang, "Fpga-accelerated maze routing kernel for vlsi designs," in *2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2022, pp. 592–597.

[24] S. Lin and M. D. F. Wong, "Superfast full-scale cpu-accelerated global routing," 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/10.1145/3508352.3549474

[25] 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.

[26] Y.-J. Chang, Y.-T. Lee, J.-R. Gao, P.-C. Wu, and T.-C. Wang, "Nthu-route 2.0: A robust global router for modern designs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 29, no. 12, pp. 1931–1944, 2010.

[27] T. Qu, Y. Lin, Z. Lu, Y. Su, and Y. Wei, "Asynchronous reinforcement learning framework for net order exploration in detailed routing," 2021, pp. 1815–1820.

[28] T.-H. Wu, A. Davoodi, and J. T. Linderoth, "Grip: Scalable 3d global routing using integer programming," in *2009 46th ACM/IEEE Design Automation Conference*, 2009, pp. 320–325.

[29] R. Agrawal, K. Ahuja, C. Hau Hoo, T. Duy Anh Nguyen, and A. Kumar, "Paralarpd: Parallel fpga router using primal-dual sub-gradient method," *Electronics*, vol. 8, no. 12, 2019. [Online]. Available: https://www.mdpi.com/2079-9292/8/12/1439

[30] P. Yao, P. Zhang, and W. Zhu, "Pathfinding model and lagrangian-based global routing," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*, 2023, pp. 1–6.

[31] E. Shragowitz and S. Keel, "A global router based on a multicommodity flow model," *Integration*, vol. 5, no. 1, pp. 3–16, 1987. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167926087800032

[32] C.-H. Hsu, H.-Y. Chen, and Y.-W. Chang, "Multi-layer global routing considering via and wire capacities," in *2008 IEEE/ACM International Conference on Computer-Aided Design*, 2008, pp. 350–355.

[33] C. Chu and Y.-C. Wong, "FLUTE: Fast lookup table based rectilinear steiner minimal tree algorithm for VLSI design," vol. 27, no. 1, pp. 70–83, 2008.

[34] M. L. Fisher, "An applications oriented guide to lagrangian relaxation," vol. 15, no. 2, pp. 10–21, 1985.

[35] R. Agrawal, K. Ahuja, D. Maheshwari, M. U. Shaikh, M. Bouaziz, and A. Kumar, "Parallel fpga routers with lagrange relaxation," *IEEE Access*, vol. 11, pp. 121 786–121 799, 2023.

[36] P. Tseng and D. P. Bertsekas, "On the convergence of the exponential multiplier method for convex programming," *Mathematical Programming*, vol. 60, pp. 1–19, 1993. [Online]. Available: https://api.semanticscholar.org/CorpusID:13946216

[37] B. W. Kort and D. P. Bertsekas, "A new penalty function method for constrained minimization," in *Proceedings of the 1972 IEEE Conference on Decision and Control and 11th Symposium on Adaptive Processes*, 1972, pp. 162–166.

[38] D. P. Bertsekas, *Nonlinear programming*. Athena scientific Belmont, 1999.

[39] T.-W. Huang, D.-L. Lin, C.-X. Lin, and Y. Lin, "Taskflow: A lightweight parallel and heterogeneous task graph computing system," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 6, pp. 1303–1320, 2022.