Top-Level Routing for Multiply-Instantiated Blocks with Topology Hashing

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

ABSTRACT

Modern System-on-Chip (SoC) design is divided into hierarchical instances using the multiply-instantiated block (MIB) technique to simplify the design process. Top-level routing aims at providing routing prototyping between those instances. It requires consideration of replicated routing paths that can either be utilized for routing or remain as floating segments. Conventional path-searching based algorithm often fails to find a legal solution under such a scenario. To address this, we propose an effective and efficient top-level routing framework for MIBs by hashing the topology of each net and using a group maze routing scheme. Experimental results demonstrate promising performance compared to the winners of the MIB-aware top-level router contest 2022 organized by Synopsys.

ACM Reference Format:
J. Wang, X. Jiang and Y. Lin. 2024. Top-Level Routing for Multiply-Instantiated Blocks with Topology Hashing . In *61st ACM/IEEE Design Automation Conference (DAC '24), June 23–27, 2024, San Francisco, CA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3649329.3655900

1 INTRODUCTION

As the scale and complexity of modern System-on-Chips (SoCs) have increased rapidly in recent years, it is difficult for designers to develop the whole design individually. Therefore, dividing a large design into multiple levels of blocks using the multiply-instantiated block (MIB, also called multiply instantiated modules) technique is applied to ease the implementation of the design process [17]. Different instances of the same MIB block share the same placement and routing results. Figure 1(a) shows an example of the floorplaning result of an MIB SoC Design.

Designers often need to finish routing for critical nets at the top level between MIB instances before going to the detailed implementation of them. Different from typical routing problems inside circuit blocks [5, 11, 13], the routing paths inside an instance of an MIB block will be copied into other instances of the same block, which may cause shorts between pins not supposed to connect. Besides such a path copy rule, other design rules like the crossing rule are also different from those in conventional routing. Those unique rules raise challenges for existing routing algorithms to efficiently and effectively generate a legal routing result with minimum wirelength.

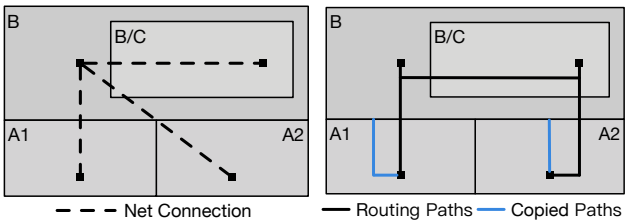

Figure 1: (a) A floorplanning result of 4 instances with a net connecting 4 pins. A1 and A2 are both instantiations of block A, and A2 is flipped by the y-axis. (b) Routing paths and paths copied by the path copy rule of the net shown in (a).

Existing routing algorithms mostly follow a negotiation-based routing scheme [1, 9, 11, 13, 14, 16]. The key idea is to model the entire layout as a grid graph and find a routing path with the least cost using shortest path algorithms [4, 7]. They iteratively resolve routing congestion with the rip-up and reroute scheme. Such approaches are likely to cause shorts when considering the path copy rule due to the greedy nature of the path-searching algorithms. Another line of research explores directly building rectilinear Steiner trees for nets [2, 3], while most of them can neither consider path copy nor routing conflicts between nets. A few studies [8, 18] formulate routing problems into integer linear programming (ILP) to search for routing paths of multiple nets simultaneously. However, solving an ILP problem is time-consuming and not scalable with problem sizes.

In this work, we propose an MIB-aware top-level router with a net topology hashing technique to help our router precisely estimate the routing cost corresponding to the path copy rule. To efficiently and effectively handle the design rule violations caused by the path copy rule and the crossing rule, we regard the original 2-D MIB SoC design as a 3-D double-layer sparse grid graph and use a group maze routing algorithm to generate the routing solution. The major contributions of this paper are summarized as follows.

- We propose a robust MIB-aware top-level router to efficiently and effectively generate top-level routing solutions between different instances of MIBs.
- We propose a net topology hashing technique to reduce the wirelength for those nets with the same topology.

*Corresponding author

DAC '24, June 23–27, 2024, San Francisco, CA, USA

ACM ISBN 979-8-4007-0601-1/24/06…$15.00
https://doi.org/10.1145/3649329.3655900

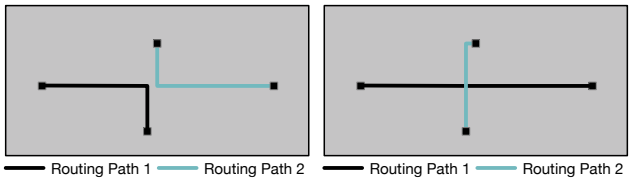

Figure 2: (a) Path 1 and path 2 violate the parallel-run spacing rule since the parallel paths of them are overlapped. (b) Path 1 and path 2 do not violate the design rule constraints.

- We propose a group maze routing algorithm on a double-layer sparse grid graph to handle the path copy rule and the crossing rule efficiently.
- Comparing to the contest winners of the MIB-aware top-level router contest 2022 organized by Synopsys (a problem in the 2022 Integrated Circuit EDA Elite Challenge [12]), our router achieves a 12%, 32%, and 10% smaller wirelength with high scalability and robustness.

The rest of the paper is organized as follows. Section 2 describes the design rule constraints and problem formulation of the MIB SoC top-level routing. Section 3 demonstrates the algorithm flow of our top-level router. Section 4 validates our routing algorithm with experimental results. Section 5 concludes the paper.

2 PRELIMINARIES

In this section, we introduce the background of the MIB-aware top-level routing problem and demonstrate our design rule constraints.

2.1 Background

To reduce the work of SoC designers, an SoC design can be divided into several MIBs. Each block can have several instances and each instance shares the same routing results. The boundaries of a block are horizontal or vertical, and there can be sub-blocks inside a block. An instance of a block can be flipped by the x-axis and/or y-axis. Figure 1(a) shows a floorplanning result of 4 instances of 3 blocks. In this example, instance B/C is a sub-block of instance B. Instances A1 and A2 are both instantiations of block A, and A2 is flipped by the y-axis.

There are Nets at the top level that connect different pins inside different instances. The target of the top-level routing is to find a routing path to connect each pin of a net. As shown in Figure 1(a), there is one net connect 4 instances. An example of its routing solution is shown in Figure 1(b). The routing path for a net shall be vertical or horizontal, and it shall follow the design rule constraints described in the following section.

2.2 Design rule constraints

To generate a legal routing result for each net connecting different instances, the top-level router shall follow the basic design rules as follows:

Connectivity rule and shorts rule. Each pin of a net shall be connected using Manhattan paths vertically or horizontally. Also, the routing result of a net shall not overlap with pins connected by other nets to avoid shorts.

Parallel-run spacing rule and crossing rule. Two different parallel routing paths shall maintain a minimum space between them. Also, the routing path shall maintain a space between the boundary of each instance and each pin it is not connected to. Different from traditional routing problems, two routing paths can cross with each other in the MIB top-level routing. As shown in Figure 2(a), path 1 and path 2 violate the parallel-run spacing rule since they are parallel paths and they are too close to each other. But in Figure 2(b), path 1 and path 2 are legal paths since they do not have parallel paths.

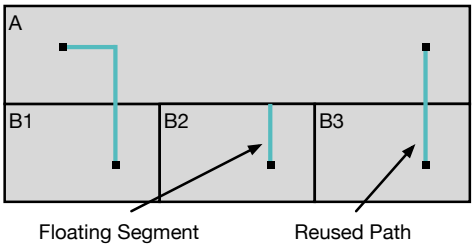

Figure 3: An example showing how the path copy rule can impact the wirelength. Paths in instance B2 and B3 are both copied from instance B1, while the path in B2 remains as a floating segment, but the path in B3 is utilized for routing.

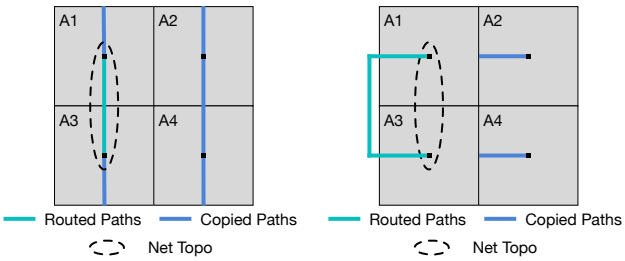

Figure 4: (a) An example showing shorts caused by the path copy rule. The copied path of the routing path connecting pins in instances A1 and A3 makes shorts occur between pins in instances A2 and A4. (b) The legal routing result of nets shown in (a).

Path copy rule. To maintain the same routing topology inside each instance of a block, the routing path inside an instance of a block will be copied into other instances of the same block. The copied path can remain as floating segments or be reused for routing. We show how the path copy rule can impact the wirelength in Figure 3. The copied path can also cause shorts between different nets, which will make the whole routing result illegal. An example of the short caused by the copied path is shown in Figure 4.

2.3 MIB-aware top-level routing problem

We formally define the MIB-aware top-level routing problem as follows. Given a set of multiple instantiated blocks B, a set of instantiations of those blocks I, a set of pins inside those instances P, and a set of nets N connecting different pins. The target is to find a legal routing result for each net following design rules described in Section 2.2 and minimize the total routed wirelength.

3 ALGORITHM

In this section, we introduce the algorithm of our MIB-aware top-level router.

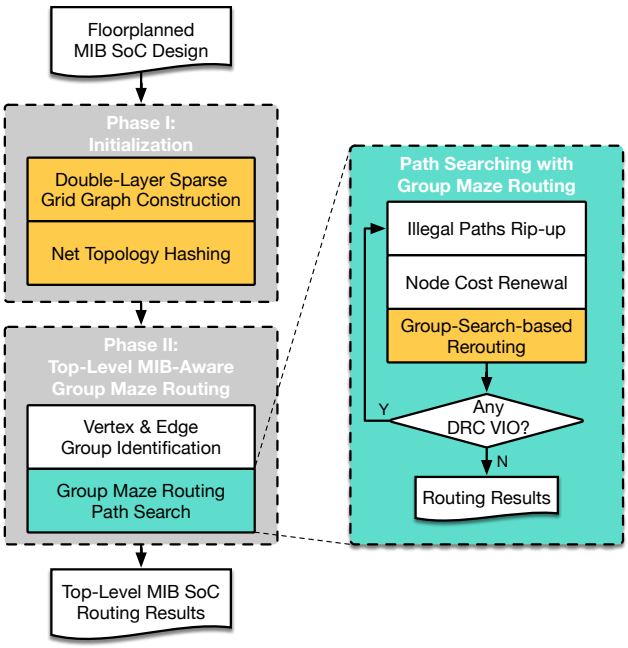

Figure 5: Overall flow of our MIB-aware top-level router.

3.1 Overall flow of our algorithm

We show the overall flow of our MIB-aware top-level router in Figure 5. Our top-level router takes the floorplanning result of each MIB instance and the net topology among pins inside those instances as input and generates the Manhattan routing path for each net as its output. There are 2 phases in our routing flow: (1) data structure initialization and (2) MIB-aware group maze routing.

Our router first builds a double-layer sparse grid graph (Section 3.2) to help our group router deal with the design rules and reduce the search space. We use the net topology hashing technique to identify the nets with the same topology and help our group maze router precisely estimate the routing cost (Section 3.3).

Traditional maze routers [10] search the routing path of a net with a single vertex in the search frontier. However, due to the greedy nature of the path-searching algorithms, those routers may find a routing result with shorts caused by the path copy rule. To avoid this problem, we maintain a group of vertices corresponding to the path copy rule in the search frontier. We describe our group maze searching method in Section 3.4.

3.2 Double-layer sparse grid graph construction

As the routing path of a net can be freely routed vertically and horizontally in an SoC layout, it brings the top-level router a large search space. The target of our double-layer sparse grid graph construction is to decrease the search space of our router by limiting the possible routing paths to several routing tracks. As shown in Figure 6(a), the tracks we generate are similar to Hanan Grids [6]. We have additional tracks to deal with the parallel-run spacing rule and the path copy rule.

Some works [20] construct a 2-D grid graph to generate routing paths. However, it is difficult for such a 2-D grid graph to check whether two routing paths are crossed with each other since the vertex representing the cross point will be captured by both the two paths under such a scenario. Therefore, we regard the 2-D

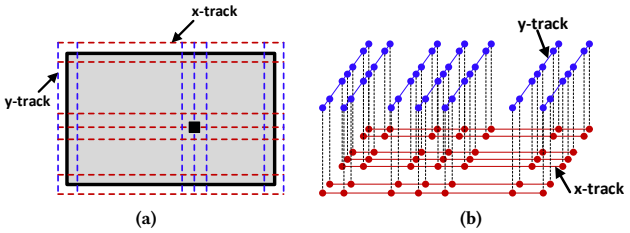

Figure 6: (a) An example of how we generate the x-tracks (red) and the y-tracks (blue) for an instance with a pin. (b) The double-layer sparse grid graph corresponds to the instance shown in (a).

layout of an SoC design as a 3-D search space and build a double-layer sparse grid graph to reduce the work of checking the design rule violations.

Our double-layer sparse grid graph is a bi-directional graph $G(V, E)$. Each vertex $v \in V$ represents (x, y, l). (x, y) indicates the position of vertex v on the routing layout, and $l \in \{0, 1\}$ represents the layer of vertex v. As shown in Figure 6(b), in layer 0, vertices are connected to their horizontal neighbors, and vertices in layer 1 are connected to their vertical neighbors. A bi-directional edge e connecting 2 vertices on the same layer has a weight w_e as the distance between positions of two vertices. It is used to represent a connection between such 2 vertices. Two vertices at the position but on different layers are also connected by a bi-directional edge. If one of those edges is used to route a net, it represents the routing paths of the net changing its direction between vertically and horizontally at the position where the edge is located. If the routing paths of two nets cross at the same location, they capture two vertices at the same location on different layers without overlapping with each other.

For any pin on the SoC layout, we define the vertex at the same position on layer 0 as its corresponding vertex on the double-layer sparse grid graph G. Therefore, we can map any net n onto G correspondingly and the target of our router is to find a routing path of net n on G between vertices representing the pins the net connects to.

Algorithm 1: Construct Double-Layer Sparse Grid Graph

Input: Position of each instance and each pin inside instances
Output: Double-layer sparse grid graph $G(V, E)$

1 Add initial x-tracks and y-tracks into track sets $\mathcal{X}$ and $\mathcal{Y}$.
2 **repeat**
3 **foreach** *newly added* $x \in \mathcal{X}$ *and* $y \in \mathcal{Y}$ **do**
4 Add new tracks generated from x or y by path copy constraint into $\mathcal{X}$ or $\mathcal{Y}$.
5 **end**
6 **until** *No new tracks are added into* $\mathcal{X}$ *or* $\mathcal{Y}$;
7 Determine vertices at 2 layers and generate edges.
8 Remove illegal edges.

We list how we construct our double-layer sparse grid graph in Algorithm 1. At first, we initialize the x-tracks $\mathcal{X}$ and y-tracks $\mathcal{Y}$ (line 1) using the coordinates of each pin, its nearest tracks satisfying the parallel-run spacing rule and the nearest tracks of boundaries of each instance satisfying the parallel-run spacing rule (Figure 6(a)). As the routing paths are copied in different

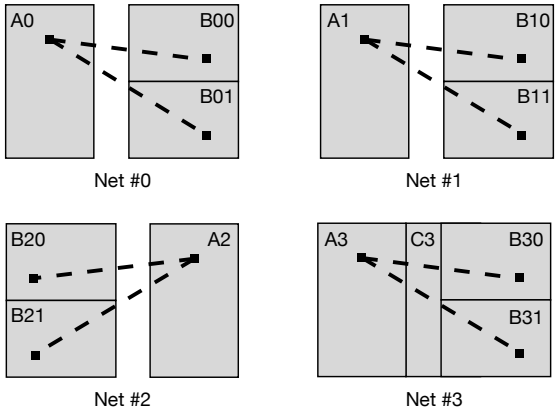

Figure 7: An example showing nets with or without the same topology. Net #0, #1, and #2 have the same topology, as Net #1 is copied from Net #0 and Net #2 is flipped by the y-axis from Net #0. Net #3 does not have the same topology as them.

instances of the same block due to the path copy rule, we iteratively insert tracks generated by path-copying into $\mathcal{X}$ and $\mathcal{Y}$ (line 2-6). With $\mathcal{X}$ and $\mathcal{Y}$ generated, we create vertices on 2 layers at each cross point of an x-track and a y-track. After vertices are created, we build edges connecting horizontal neighbors in layer 0, vertical neighbors in layer 1, and vertices representing the same location on different layers (line 7). To avoid the parallel-run spacing rule violation, we then remove edges that are too close to the boundaries of each block (line 8).

3.3 Net topology hashing

To simplify the SoC design process, nets with the same topology commonly appear in an MIB SoC design. As shown in Figure 7, net #0, #1, and #2 have the same topology, and net #3 does not have the same topology as them. To help our top-level router precisely estimate the routed cost and efficiently route the MIB SoC design, we develop our net topology hashing technique to identify nets with the same topology. We generate a hash code for each net in the MIB SoC design. Those nets with the same topology shall have the same hash code while the hash code of nets with different topology shall also be different.

We list how we use our net topology hashing technique to generate the hash code for net n in Algorithm 2. We take the position of each instance and the position of the pins net N connects to as the input. We first generate a bounding box b to cover all the instances the net connects (line 1). As there may be instances overlapping with b, and those instances can also influence the routing paths of n, we iteratively expand the boundary of b to cover them (lines 2-4). After the area of b does not increase, we record the block type and flipped directions of all the instances covered by b (line 5). As the instances can be flipped by the x-axis, y-axis, or both the x-axis and y-axis, we generate the hash code from the down-left, the down-right, the top-left, the top-right points of b regarding all the flip directions for a net (line 6).

With the hash code generated for each net, we define two nets with the same topology if their four hash codes are the same after reordering. All the nets with the same topology are defined as a net group. With the path copy rule, the routing paths of each net of the same topology shall be the same to reduce the wirelength.

Algorithm 2: Generate Hash Result for Net N

Input: Position of each instance and position of pins connected by net n
Output: Hash result of net N
1 Initialize a bounding box b covering all instances connected by N
2 **repeat**
3 | Expand b to cover every instance it overlaps with.
4 **until** *Size of b does not increase*;
5 Record all instances I covered by b and their flipped direction D.
6 Generate hash results of I, D, and pins connected by n from four directions.

This feature is used by our group maze router to precisely estimate the routing cost.

3.4 Group maze routing scheme

The target of our group maze router is to find each net a routing path with a minimum wirelength. It shall also efficiently and effectively deal with the design rules listed in Section 2.2. Traditional maze routers often fail to deal with the unique design rules in the MIB top-level routing. As shown in Figure 8, expanding the routing path to vertex v1 will cause shorts between instance B1 and B2 by the path copy rule. Traditional routers cannot deal with the design rule violation caused by such a scenario since they only consider the status of the single vertex in the frontier. To deal with this problem, our group maze routing scheme considers both the current expanding vertex and vertices in other instances with the same block type at the same offset position to estimate whether the current expanding vertex can be expanded.

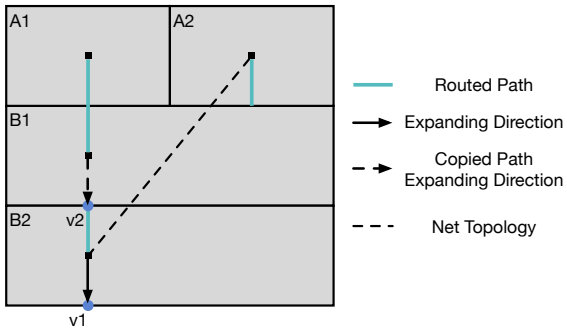

Figure 8: A scenario where the traditional path-searching method fails to find a legal solution.

As shown in Figure 5, our group maze router follows the negotiation-based [15] method to iteratively generate a routing result. Before we apply our group maze routing algorithm to find routing paths for each net, we first record vertices and edges belonging to the same group corresponding to the path copy rule. Then we iteratively rip up those nets with design rule violations and apply our group maze routing algorithm to find routing paths for those congested nets until there is no design rule violation.

To route a net using our group maze routing algorithm, we select a pin connected by the net as the source pin and start routing from its corresponding vertex on our grid graph. Other pins of the net are regarded as the target pins. We find paths from the source pin to each target pin one by one. We list how we find

routing paths to a target pin in Algorithm 3. When searching paths to a target t in net n, our router will initialize a priority queue Q and add all the routing paths of n into Q (line 1). We pop out the frontier routing path from Q iteratively. If the frontier path reaches the target vertex, we end our path-finding process and return the frontier path as the routing result (lines 3-5). Otherwise, we check each neighbor vertex v and group vertices of v of the head vertex of p. We define v and its group vertices are legal to expand if expanding to them does not violate any design rule constraint. If they are legal to expand, we estimate the routing cost of v and expand the routing path to v (lines 8-10).

Algorithm 3: Group Maze Routing

Input: Double-layer sparse grid graph G, net n needs to be routed, routing target t
Output: Manhattan routing path p to t

```
1  Add all of the vertices and their group vertices used to
     route n into a priority queue Q.
2  while Q is not empty do
3      Pop the frontier of Q as p.
4      if p reaches t then
5          Return p.
6      end
7      foreach neighbor vertex v of p do
8          if v and its group vertices are legal to expand then
9              Calculate the routing cost of expanding to v.
10             Add v to Q.
11         end
12     end
13 end
```

We use the following function to estimate the routing cost c_v of grid vertex v when expanding the routing path from an unused grid edge e to route net n:

$$c_v = \begin{cases} (w_e + p_v) * \dfrac{g_e}{g_n}, & e \in I \\ w_e + p_v, & e \notin I \end{cases} \quad (1)$$

w_e is the initial weight of edge e. For edges connecting vertices on the same layer, it equals the distance between two vertices. For edges connecting different layers, it is defined as a positive small number w_d. p_v is the penalty cost of vertex v and it includes 2 parts h_v and b_v. h_v refers to the historical congestion cost and is updated after each rip-up and reroute iteration. b_v refers to the out-of-area cost and it is used to help nets with multiple pins find routing paths with smaller wirelength. g_e refers to the edge group size of edge e, and g_n refers to the number of nets with the same topology with net n. Considering the path copy rule, we multiply the cost by $\frac{g_e}{g_n}$ for those edges inside an instance I. Edges outside of instances are not influenced by the path copy rule. Their routing cost will be estimated as the sum of their initial weight and the penalty cost of v.

4 EXPERIMENTAL RESULTS

We implement our algorithm in C++ and conducted experiments on a Linux machine equipped with an AMD EPYC 7542 32-Core Processor (2.90 GHz) and 384 GB RAM. We collect the 5 public cases from the contest [12] provided by Synopsys (P-Cases) and generate 5 cases following the guide from the contest organizers (G-Cases) as our benchmarks. We show the statistics of our benchmarks in Table 1. There can be hundreds of instances and nets in our benchmarks.

Table 1: The statistics of out benchmarks.

Design	#Blocks	#Insts	#Pins	#Nets	#Conns
P-Case #0	8	12	16	4	4
P-Case #1	2	64	64	16	16
P-Case #2	72	108	108	36	36
P-Case #3	3	180	180	36	72
P-Case #4	2	100	100	50	50
G-Case #0	3	400	400	100	300
G-Case #1	3	216	252	108	108
G-Case #2	3	448	448	128	256
G-Case #3	1	225	225	50	50
G-Case #4	4	588	392	98	196

We obtain the binary files of top-3 teams winning the Synopsys Special Award of the contest [12]. We run their routers and our router on the 10 test cases and list the result of the runtime, the wirelength, and the number of design rule violations on Table 2. Note that [19, 20] also solve the MIB-aware top-level routing problem using a negotiation-based algorithm and a Steiner-tree-based algorithm. We have contacted the authors but learned that neither their binary nor benchmarks are available, so we cannot make a comparison with their methods (note their team does not enter the top-3 teams of the contest). The result shows that in the public cases, within similar runtime, our router can acquire a 12%, 32%, and 10% smaller wirelength compared to the contest winners on average. In our generated cases, our router has a high scalability and high robustness. We pass all 5 test cases while the contest winners pass at most 2 cases. In those cases passed by the router of the contest winners, our router can achieve up to 67% smaller wirelength.

To validate the effectiveness of our net-hashing technique, we run our router on the 10 benchmarks with and without net hashing. When running without net hashing, g_n in Equation 1 for each net is always 1 to estimate the routing cost. The comparison between running with and without net hashing is shown in Table 3. It shows that our net-hashing technique can help achieve a 12% smaller wirelength on the public cases of the contest [12]. In our generated cases, routing without the net-hashing technique fails to generate a legal solution within the maximum rip-up and reroute iterations on G-Case #2, and it takes 84% more runtime to generate routing results with 37% larger wirelength on the other 4 benchmarks compared to routing with the net-hashing technique.

Figure 9 shows the runtime breakdown of our top-level router on G-Case #4. The process of our double-layer sparse grid graph construction takes 61.24% of the total runtime. Doing our group maze routing search takes 28.48% of the total runtime. Those 2 processes are the major portion of the total runtime. Generating hash code for each net and Identifying vertices and edge groups only takes 4.80% of the total runtime, and they can effectively improve the quality of the final routing result. The other part of runtime is used to read and write files, which takes 5.48% runtime of total flow.

5 CONCLUSION

In this paper, we propose an MIB-aware top-level router to generate routing results for the MIB SoC design after floorplanning. We

Table 2: Routed wirelength (WL), routing runtime (RT, ms), and the number of design rule violations (#DRVs) comparison between the contest winners [12] and our router.

Design	Team 1			Team 2			Team 3			Ours		
	WL	RT	#DRVs	WL	RT	#DRVs	WL	RT	#DRVs	WL	RT	#DRVs
P-Case #0	**1064**	6	0	1440	5	0	**1064**	4	0	**1064**	60	0
P-Case #1	8096	25	0	**5760**	13	0	**5760**	7	0	**5760**	42	0
P-Case #2	**12996**	38	0	18720	353	0	**12996**	33	0	**12996**	100	0
P-Case #3	23624	242	0	31320	6161	0	23616	164	0	**23550**	606	0
P-Case #4	22450	61	0	27580	673	0	28800	70	0	**19000**	78	0
P-Ratio	1.12	0.45	—	1.32	4.54	—	1.10	0.35	—	**1.00**	1.00	—
G-Case #0	57097	70823	0	34500	559	0	N/A	101	N/A	**27000**	1095	0
G-Case #1	N/A	183	N/A	ILLEGAL	5163207	98	N/A	94	N/A	**17934**	374	0
G-Case #2	58176	32040	0	ILLEGAL	17286	192	N/A	94	N/A	**48768**	1756	0
G-Case #3	N/A	6766	N/A	N/A	>5h	N/A	ILLEGAL	517	25	**63500**	427	0
G-Case #4	ILLEGAL	99556	49	ILLEGAL	871625	196	N/A	88	N/A	**108486**	27229	0
G-Ratio*	1.67	20.58	-	1.28	>11200.46	-	-	-	-	**1.00**	1.00	-

"ILLEGAL" means that the router provides a routing result, but the routing solution violates the design rules.
"N/A" means that the router fails to output a routing result.
*Only the wirelength of the legal routing results is counted in G-Ratio.

Table 3: Routed wirelength (WL) and routing runtime (RT, ms) comparison between routing without net hashing technique and with net hashing technique.

Design	w/o Net Hashing		w/ Net Hashing	
	WL	RT	WL	RT
P-Case #0	**1064**	**38**	**1064**	60
P-Case #1	8096	52	**5760**	**42**
P-Case #2	**12996**	108	**12996**	**100**
P-Case #3	23954	**407**	**23436**	606
P-Case #4	22450	119	**19000**	**78**
P-Ratio	1.12	1.03	**1.00**	**1.00**
G-Case #0	58000	1878	**27000**	**1095**
G-Case #1	17934	702	**17934**	**374**
G-Case #2	N/A	5286	**48768**	**1756**
G-Case #3	65000	549	**63500**	**427**
G-Case #4	140091	67187	**108486**	**27229**
G-Ratio	1.37	1.84	**1.00**	**1.00**

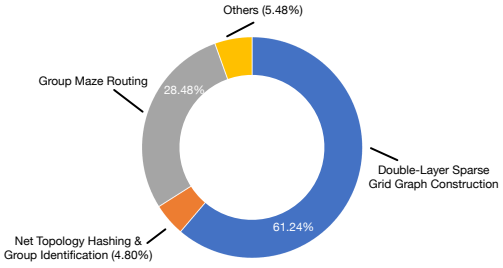

Figure 9: Runtime breakdown on design G-Case #4.

maintain the hash code for each net to identify those nets with the same routing topology, which helps our group maze router to precisely estimate the routing cost. We propose a group maze routing algorithm on a double-layer sparse grid graph to deal with the path copy rule and crossing rule efficiently. Compared to the contest winners, our router can archieve 12%, 32%, and 10% smaller wirelength with high scalability and robustness.

ACKNOWLEDGE

This project is supported in part by the Natural Science Foundation of Beijing, China (Grant No. Z230002), the National Natural Science Foundation of China (Grant No. 62034007 and 62141404), and the 111 Project (B18001).

REFERENCES

[1] Yen-Jung Chang et al. 2010. NTHU-Route 2.0: A Robust Global Router for Modern Designs. *IEEE TCAD* 29, 12 (2010), 1931–1944.
[2] Gengjie Chen et al. 2017. SALT: Provably good routing topology by a novel steiner shallow-light tree algorithm. In *Proc. ICCAD*. 569–576.
[3] C. Chu. 2004. FLUTE: fast lookup table based wirelength estimation technique. In *Proc. ICCAD*. 696–701.
[4] Edsger W Dijkstra. 1959. A note on two problems in connexion with graphs. *Numerische mathematik* 1, 1 (1959), 269–271.
[5] Michael Gester et al. 2013. BonnRoute: Algorithms and Data Structures for Fast and Good VLSI Routing. *ACM TODAES* 18, 2, Article 32 (April 2013), 24 pages.
[6] M. Hanan. 1966. On Steiner's Problem with Rectilinear Distance. *SIAM J. Appl. Math.* 14, 2 (mar 1966), 255–265.
[7] Peter E. Hart et al. 1968. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics* 4, 2 (1968), 100–107.
[8] Andrew B. Kahng et al. 2018. TritonRoute: an initial detailed router for advanced VLSI technologies. In *Proc. ICCAD*. ACM, 1–8.
[9] Andrew B. Kahng et al. 2021. TritonRoute: The Open-Source Detailed Router. *IEEE TCAD* 40, 3 (2021), 547–559.
[10] C. Y. Lee. 1961. An Algorithm for Path Connections and Its Applications. *IRE Transactions on Electronic Computers* EC-10, 3 (1961), 346–365.
[11] Haocheng Li et al. 2019. Dr. CU 2.0: A Scalable Detailed Routing Framework with Correct-by-Construction Design Rule Satisfaction. In *Proc. ICCAD*. 1–7.
[12] Bohai Liu et al. 2022. Multiply instantiated block-aware top-level router. https://eda.icisc.cn/en/file/cacheFile/ee280e8ccf5647ff987ea0e9fc6d9357.pdf.
[13] Jinwei Liu et al. 2020. CUGR: Detailed-Routability-Driven 3D Global Routing with Probabilistic Resource Model. In *Proc. DAC*. 1–6.
[14] Wen-Hao Liu et al. 2013. NCTU-GR 2.0: Multithreaded Collision-Aware Global Routing With Bounded-Length Maze Routing. *IEEE TCAD* 32, 5 (2013), 709–722.
[15] L. McMurchie and C. Ebeling. 1995. PathFinder: A Negotiation-Based Performance-Driven Router for FPGAs. In *Proc. FPGA*. 111–117.
[16] Jarrod A. Roy and Igor L. Markov. 2008. High-Performance Routing at the Nanometer Scale. *IEEE TCAD* 27, 6 (2008), 1066–1077.
[17] Synopsys. 2023. Smarter System-on-Chip Signoff with Multiply Instantiated Module Support. https://www.synopsys.com/implementation-and-signoff/resources/videos/smarter-soc-signoff.html.
[18] Jiarui Wang et al. 2023. A Robust FPGA Router with Concurrent Intra-CLB Rerouting. In *Proc. ASPDAC (ASPDAC '23)*. Association for Computing Machinery, New York, NY, USA, 529–534.
[19] Hang Yang et al. 2023. Multi-instantiated Block Top-layer Routing Technique Based on Steiner Tree Algorithm. In *Proc. ISEDA*. 274–279.
[20] Hang Yang et al. 2023. Multi-Instantiation Top-Level Routing Technique Based on Decision Negotiation Algorithm. In *Proc. ICET*. 629–634.