SPRoute 2.0: A detailed-routability-driven deterministic parallel global router with soft capacity

Jiayuan He
Computer Science, UT Austin
Austin, TX, USA
hejy@cs.utexas.edu

Udit Agarwal
Katana Graph
Austin, TX, USA
udit@katanagraph.com

Yihang Yang
Electrical Engineering, Yale University
New Haven, CT, USA
yihang.yang@yale.edu

Rajit Manohar
Electrical Engineering, Yale University
New Haven, CT, USA
rajit.manohar@yale.edu

Keshav Pingali
Computer Science, UT Austin
Austin, TX, USA
pingali@cs.utexas.edu

2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC) | 978-1-6654-2135-5/22/$31.00 ©2022 IEEE | DOI: 10.1109/ASP-DAC52403.2022.9712557

***Abstract*—Global routing has become more challenging due to advancements in the technology node and the ever-increasing size of chips. Global routing needs to generate routing guides such that (1) routability of detailed routing is considered and (2) the routing is deterministic and fast. In this paper, we firstly introduce soft capacity which reserves routing space for detailed routing based on the pin density and Rectangular Uniform wire Density (RUDY). Second, we propose a deterministic parallelization approach that partitions the netlist into batches and then bulk-synchronously maze-routes a single batch of nets. The advantage of this approach is that it guarantees determinacy without requiring the nets running in parallel to be disjoint, thus guaranteeing scalability. We then design a scheduler that mitigates the load imbalance and livelock issues in this bulk synchronous execution model. We implement *SPRoute 2.0* with the proposed methodology. The experimental results show that SPRoute 2.0 generates good quality of results with 43% fewer shorts, 14% fewer DRCs and a 7.4X speedup over a state-of-the-art global router on the ICCAD2019 contest benchmarks.**

***Index Terms*—global routing, detailed routability, parallelization, determinism**

I. Introduction

The routing problem in VLSI chip design is usually solved in two steps known as *global routing* and *detailed routing*. Global routing partitions the 3D routing space into regular rectangles (GCells) and generates a 3D grid graph in which each node represents a GCell and each edge represents the connection between adjacent GCells. The goal of global routing is to generate a routing solution on the grid graph and provide a preferred routing region (Guide) for the detailed router. The global routing solution should maximize routability for the detailed router to manage design rule checks (DRCs), pin accessibility, irregular shapes, etc.

In the traditional global routing formulation, the capacity of an edge is generally set as the number of available tracks. However, consuming all given capacity of an edge in global routing may result in limited space for the detailed router to handle some complicated scenarios, such as DRCs, pin accessing, track switching, etc. Therefore, we introduce a *soft* capacity which reserves space for the detailed router to be used for complicated scenarios. The soft capacity is downsized from the hard capacity (number of available tracks), using the pin density and RUDY value of the region. With the space reservation of soft capacity, SPRoute 2.0 generates 43% fewer shorts and 14% fewer DRCs compared with a state-of-the-art global router.

In terms of parallelization, maze routing is widely used in global routing and is the most time-consuming stage on hard-to-route benchmarks. A common strategy to parallelize maze routing is net-level parallelization, i.e. each thread acquires a net and applies maze routing on its local grid graph. Threads may route nets through the same region concurrently, thus resulting in a race condition and non-deterministic solution. A deterministic parallel approach is to route nets with disjointed routing regions concurrently. But the parallelism of this approach is limited due to the large overlapped region of different nets. To achieve good speedup, some parallel global routers like NCTU-gr [1] and SPRoute [2] adopt non-deterministic net-level parallelization which does not enforce disjointness of concurrent nets.

We propose a *deterministic* net-level parallelization strategy, which divides the entire netlist into batches and executes batches of nets in a bulk synchronous way. The nets in a given batch are ripped up and re-routed based on the usage of the grid graph after the completion of the previous batch, and the new usage changes of the global grid graph are not applied until all the nets in the current batch are completed. In this way, nets in a given batch make a deterministic maze routing decision, irrespective of the order of execution, and thus region disjointness is not required to guarantee determinism.

There are two major issues in the performance of this bulk synchronous parallelization: load imbalance and livelock. First, threads executing a batch may have load imbalance due to the considerably different sizes of nets. Second, since nets make decisions based on the stale usages after the completion of the previous batch, nets in the same batch can compete for the same routing resources repeatedly and result in livelock. We design a scheduler which reduces the impact of these two

978-1-6654-2135-5/22/$31.00 ©2022 IEEE

issues and achieves a good speedup.

The contributions of this paper are summarized as follows:

- We propose a soft capacity to reserve space for detailed routability. The soft capacity is adjusted from the hard capacity based on the pin density and RUDY of the neighboring GCells.
- We parallelize maze routing in a deterministic bulk synchronous approach which does not require net disjointness to guarantee determinism.
- We design a scheduler for the deterministic parallel execution model to mitigate the performance impact of load imbalance and live lock.
- We implement *SPRoute 2.0* based on the proposed methodology. Compared with the state-of-the-art, our global router achieves better quality of results with fewer shorts and DRCs and a significant speedup.

The rest of the paper is organized as follows: Section II describes the preliminaries and the relevant background for the global routing problem. Section III provides a brief overview of the related work in this area. In Section IV, we describe the proposed soft capacity and deterministic maze routing, followed by the experimental results in Section V. Finally, we conclude our work in Secion VI.

II. Preliminaries

A. Global Routing

In the traditional global routing formulation, the 3D routing space is partitioned into regular rectangles (GCells), resulting in a 3D grid graph $G(V, E)$, where each vertex $v \in V$ denotes a GCell and each $e \in E$ denotes the connection between adjacent GCells. The capacity of an edge represents the maximum number of wires that can go through the edge. The usage of an edge is the number of wires that are currently using the edge and the overflow is denoted by the number of wires that exceeds the capacity.

The global routing problem is, for each net, to find a path that connects all the pins of a net in the given grid graph without any overflow on the edges, minimizing the total wirelength and number of vias.

B. Evaluation Metrics

Traditional global routing solutions are evaluated by a combination of the total overflow, total wirelength and the number of vias. In the ICCAD-2019 contest [3], the global routing solution is evaluated by a state-of-the-art detailed router Dr.CU [4]. Dr.CU first reads the routing guides as input and performs detailed routing. Then Cadence Innovus [5] is used to extract the quality information based on the metrics in Table I, which includes basic usage (wire length and the number of vias), non-preferred resource usage and DRC violations.

C. RUDY

Rectangular Uniform wire Density (RUDY) [6] estimates the wirelength density by uniformly spreading the wire volume of nets into its bounding box. It is commonly used as a feature to indicate the congestion of a region in several routability prediction works. RUDY of a net represents the average wire length per unit area in the bounding box of the net, and is computed by:

TABLE I
Metrics in ICCAD-2019 contest

Category	Metric	Weight
basic usage	length of wire	0.5
	number of vias	4
non-preferred usage	length of wrong-way wire	1
	number of off-track vias	1
	length of off-track wires	0.5
	length of out-of-guide wires	1
	number of out-of-guide vias	1
DRC violations	number of min-area violations	500
	number of spacing violations	500
	number of short violations	500
	short metal area / M2 pitch	500

$$RUDY_n(x,y) = \begin{cases} \frac{wire\ length}{bbox\ area} & x \in [x_{min}, x_{max}], y \in [y_{min}, y_{max}] \\ 0 & otherwise \end{cases} \tag{1}$$

Here the x_{min} and x_{max} represents the maximum and minimum x-axis coordinates of the net, and y_{min} and y_{max} represents the maximum and minimum y-axis coordinates of the net. The RUDY of a location (x, y) is computed by aggregating RUDY of all nets in netlist N:

$$RUDY_N(x,y) = \sum_{n \in N} RUDY_n(x,y) \tag{2}$$

III. Related Work

A. Global Routing and Detailed Routability

Most global routers including FastRoute 4.0 [7], NCTU-GR 2.0 [1], NTHU-Route [8], NTUgr [9] focus on generating routing paths on a grid graph with given edge capacity and do not consider detailed routability. VFGR [10] introduces an accurate congestion model for several layout components like fat vias, and proposes a pass-through capacity on the node. CUGR [11] works directly on the 3D grid graph and proposes a probabilistic resource model for detailed routability. Taghavi et al. [12] use pin geometries and density to measure detailed routability in routability-driven placement. GLARE [13] utilizes pin density for better congestion analysis.

We choose CUGR as our baseline because (1) it is one of the available state-of-the-art global routers and (2) the other congestion models are not readily evaluated in the flow since they are not implemented in a global router.

B. Parallelization in Maze Routing

Global routers have adopted a number of parallelization techniques. PGRIP [14], a parallel version of GRIP [15], utilizes integer programming and processes routing subproblems corresponding to rectangular subregions covering the chip area. Han [16] implement net-level parallelization on GPUs by identifying and scheduling independent nets. VFGR [10] utilizes both net-level and region-level parallelization. NCTU-GR 2.0 [1] studies race conditions on routing resources and introduces a collision-aware rip-up and reroute solution by

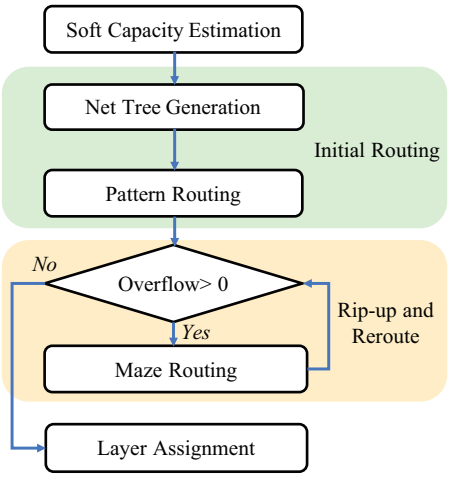


Fig. 1. Overall flow of SPRoute 2.0

adjusting the cost of conflicting routing resources. SPRoute [2] introduces a two-phase parallel scheme to address the livelock issue in net-level parallelization.

IV. Proposed Algorithm

A. Overall flow

Fig. 1 shows the overall flow of SPRoute 2.0. Firstly, the soft capacity is computed according to the pin density and RUDY of the region, then net tree generation and pattern routing are executed to generate an initial routing solution. If there is overflow in the solution, the global router iteratively performs maze routing using the proposed deterministic parallel methodology. Finally when overflow decreases to zero or the maximum number of iterations is reached, layer assignment is executed to generate 3D routing guides. The overflow computed in SPRoute 2.0 is based on the soft capacity to reserve space for detailed routability.

B. Soft Capacity and Cost function

1) DRCs and congestion: Fig. 2 (a) shows the screenshot of the whole chip of benchmark em ispd19_test7. The grey region in the middle left is the standard cell region and the rectangles around it are memory blocks. The yellow bar at the edge of the chip are I/O pins. Fig. 2 (b) and (c) shows the pin density map and RUDY map where the majority of the congestion is in the standard cell region. The bottom four figures (e)-(h) show the DRC map from Metal2 to Metal5 if the capacity of all edge reserves 10% for the detailed routabilty, i.e. soft capacity = 90% of the number of available tracks. These maps demonstrate that the DRC location is related to the pin density and the RUDY of the region, and high metal layers are less affected by pin density and RUDY than lower metal layers.

Based on the relationship among pin density, RUDY and DRC map, we estimate the congestion of a region by adding the pin density and RUDY:

$$cong\left(x,y\right) = pin\_density\left(x,y\right) + w * RUDY\left(x,y\right) \tag{3}$$

where w is a constant weight. The $RUDY$ is computed by the half perimeter wire length (HPWL) and the area of the bounding box.

2) Soft Capacity: The soft capacity of an edge indicates the routing resources after reserving space for detailed routing and is used to compute the overflow for the termination of rip-up and reroute. The soft capacity is adjusted from the hard capacity based on a $ratio$ function:

$$soft\_cap\left(x,y\right) = ratio\left(\overline{cong}\left(x,y\right)\right) * hard\_cap\left(x,y\right) \tag{4}$$

where $hard\_cap$ is the number of available tracks between GCells, i.e. total number of tracks minus the number of tracks consumed by blockages or pins. The $ratio$ function is a logistic function of the congestion $\overline{cong}(x,y)$, which takes the mean of the $cong(x,y)$ of adjacent GCells. Fig. 2 (d) gives an example of the ratio function:

$$ratio\left(cong\right) = min + \frac{max-min}{1+exp((cong-cong_{mid})*k)} \tag{5}$$

where min and max denotes the minimum and maximum value of the ratio function, k is the slope which determines the sensitivity to the congestion, and $cong_{mid}$ is the midpoint of the logistic function.

In the ratio function shown in Fig. 2 (d), if $cong$ is low, the soft capacity is about 90% of the hard capacity and does not drop much as congestion increases, since a small amount of congestion does not affect the routability significantly. If $cong$ is close to $cong_{mid}$, the soft capacity decreases dramatically as congestion increases. If $cong$ is high, the ratio function reaches a bottom line since the edge between GCells still needs to maintain capacity for possible connectivity.

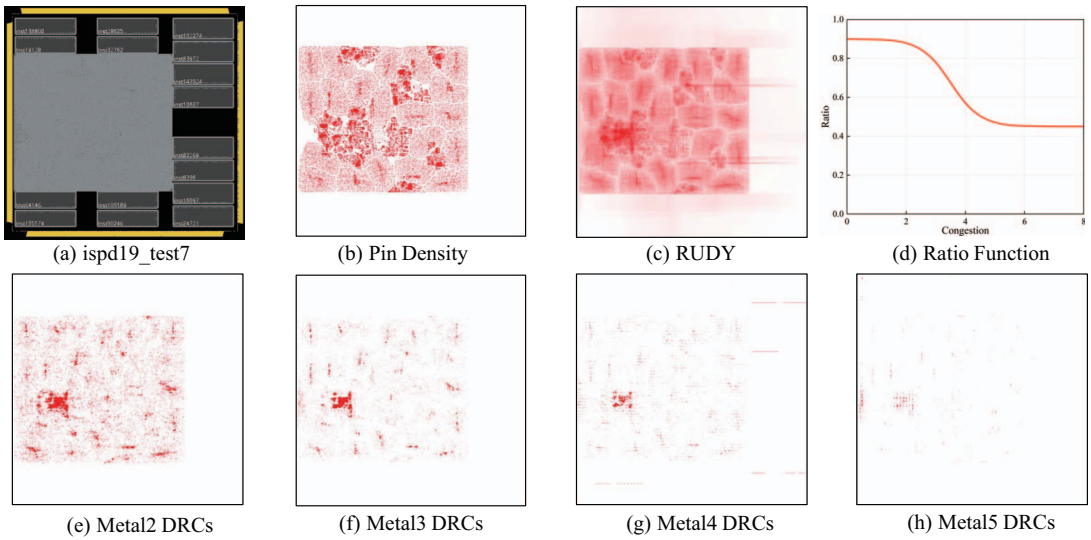
Different layers have different parameters for the ratio function since they are influenced by the congestion in different scales. As shown in Fig 2(e)-(h), low metal layers have more DRCs than high metal layers thus require more space reservation. This is because 1) low layers are affected by pin accessibility more than high layers and 2) generally low layers are used for local nets which consists of turns and vias, while high layers are used for long-range wires. Long-range parallel wires are regular and do not require much space reservation. Therefore, different parameters are used in different layers in our global router, and these parameters are uniform for all benchmarks.

C. Initial Routing

A common strategy to route a multi-pin net is to first decompose it into many two-pin edges to generate a routing tree, and then route each two-pin edge individually. In Fig. 1, we utilize FLUTE [17] to generate a rectilinear Steiner minimum tree (RSMT) for each net and then apply pattern routing techniques from FastRoute 4.0 [7] to generate an initial solution for rip-up and reroute.

D. Rip-up and Reroute

The rip-up and reroute stage (Fig. 1) goes over the netlist and tries to generate a better path for each net. We call the process of performing rip-up and reroute on the whole netlist as an iteration. The iterative loop terminates when overflow decreases to zero or the maximum number of iteration is reached.



(a) ispd19_test7 (b) Pin Density (c) RUDY (d) Ratio Function

(e) Metal2 DRCs (f) Metal3 DRCs (g) Metal4 DRCs (h) Metal5 DRCs

Fig. 2. Pin density, RUDY and DRCs map on ispd19_test7

Algorithm 1: Parallelization scheduler

Input: overflowing netlist N, iteration number $iter$, batch size s, total overflow tof

Output: vector of net batches B, batch size for next iteration s

1 $nbatch = \lceil N.size/s \rceil$;
2 **if** $iter = 1$ **then**
3 **if** $n.bbox > bbox\_thold$ **then**
4 **foreach** *net* $n \in N$ **do**
5 $B[n.id \ \% \ nbatch].push(n)$
6 **else if** $tof > of\_thold$ **then**
7 **if** $iter\%2 = 0$ **then** $sort\_by\_overflow\_edge\_X(N)$
8 **else** $sort\_by\_overflow\_edge\_Y(N)$
9 **foreach** *net* $n \in N$ **do**
10 $B[n.sorted\_rank \ \% \ nbatch].push(n)$
11 **else** $B[n.id \ \% \ nbatch].push(n)$
12 **if** $s >= 2 * nthreads$ **then** $s = \lceil s/2 \rceil$

1) Bulk Synchronous Parallelization: In rip-up and reroute, each net first checks whether its path contains overflowing edges or not. If so, the path is removed by decrementing the usage on all the used edges, and then rerouted to find a new path with the minimum cost. The new path is added by incrementing the usage on all the new edges.

We propose a bulk synchronous deterministic approach to parallelize rip-up and reroute. The netlist is partitioned into batches and all threads execute one batch of nets at a time. At the beginning of the batch, each thread acquires a net from the batch and reads the usage of the global graph and performs rip-up and reroute in its thread-local graph. The changes of usage of a net are not updated to the global graph immediately, but pushed into a buffer which stores all the changes of this batch. After all nets of a batch finish their rip-up and reroute, the changes in the buffer are pushed to the global graph. In this bulk synchronous model, nets read the same usage and makes the same path searching decisions regardless of the execution order in the batch, and generate deterministic solutions.

Our parallel maze routing is implemented in the Galois system [18] which is an open-source C++ library designed to ease the implementation of parallel graph algorithms. We utilize the *do_all* parallel loop in Galois to exploit net-level parallelism. The *do_all* construct partitions the batch evenly among all threads. Each thread maze routes its local partition of nets on its local grid graph and pushes changes to the buffer. We enable the work stealing functionality in Galois to reduce load imbalance.

2) Load imbalance: In our rip-up and reroute implementation, we mark a net as $finished$ once it has no overflow in an iteration. A $finished$ net will not be ripped up in future iterations even if it has new overflows caused by other nets. After the first iteration of maze routing, most small nets are marked as $finished$ and will not be executed in the later iterations. Because of this, the load imbalance issue is significant especially in the first iteration. We design a scheduler to filter out small nets in the first iteration to reduce load imbalance.

3) Livelock: The second issue of our proposed parallelization is livelock. Since nets in the same batch are not disjoint, they might compete for the same routing resources, resulting in livelock. Livelock can delay the convergence of overflow and result in long runtime, or even prevent overflow from decreasing to zero. The probability of livelock is affected by many factors including the degree of overlap of the nets, the size of the batch and the probability of the same scheduling. In our scheduler design, we distribute close nets to different batches and reduce the size of batches over iterations to reduce the probability of livelock.

4) Scheduler: At the beginning of each iteration of rip-up and reroute, $finished$ nets are firstly filtered out and then the scheduler shown in Algorithm 1 is called to partition the unfinished nets into batches. Lines 2-5 shows the scheduling to reduce load imbalance in the first iteration. The scheduler filters out nets smaller than a threshold and only pushes larger nets into batches. This reduces the difference of the net size

and load imbalance in the first iteration. The reason we filter out small nets rather than large nets is that small nets should have high priority in using local routing resources and should not detour for large nets, which has more routing flexibility.

Line 6-10 shows the net scheduling to reduce livelock after the first iteration when there is a substantial amount of overflow. Line 7-8 are two functions to sort the overflowing netlist according to their overflow edge. The sort key is the X or Y coordinate of the middle point of an overflowing 2-pin edge of a net and represents the approximate location of the routing region. The scheduler alternatively uses X or Y coordinate to sorts. If a net has multiple overflowing 2-pin edges, the scheduler picks a random one. Line 9-10 distribute close nets into different batches to reduce the overlap region and probabilty of livelock. Finally, a simple random scheduling is utilized when maze routing is close to convergence on line 11. On lines 12, the batch size is reduced by half in each iteration to reduce the probability of livelock.

V. Experimental Results

SPRoute 2.0 is implemented in C++ and the experiments are conducted on a 64-bit linux machine with 2.2 GHz Intel Xeon Gold and 196 GB of memory. We use the benchmarks and the evaluation methodology from the ICCAD2019 contest [3]. Firstly the global routing solutions are read as routing guide by Dr.CU [4] to generate a detailed routing solution, then Cadence Innovus [5] is used to examine DRCs, short, etc. and report the final scores. The score contains basic usage of wire length and vias, non-preferred usage such as wrong-way wire/vias, DRCs and shorts, as shown in Table I. Note that the ICCAD-2019 benchmarks consist of benchmarks from both ISPD-2018 and ISPD-2019, but the evaluation metrics of ISPD-2018 and ISPD-2019 are slightly different. To make our evaluation uniform and in accordance with the metric shown in Table I, we use the ISPD-2019's evaluator for all benchmarks.

A. Comparison with the State-of-the-art

We compare the quality and runtime with the state-of-the-art global router CUGR [11] in Table II. The benchmark names without $\_metal5$ are designs with nine metal layers and the ones with $\_metal5$ are the same designs with five metal layers.

We compute the geomean of the ratios over CUGR on all benchmarks and summarize the result at the bottom of Table II. The speedups are the reciprocals of the runtime ratios. SPRoute 2.0 produces quality scores close to CUGR with 0.1% degradation, and for 8 of 12 benchmarks, our solution is better. SPRoute 2.0 is 6.9X faster on single thread and 7.4 faster with 8 threads.

Table III lists the details of the quality under different metrics. The bottom ratio demonstrates the average quality compared with CUGR. Our global router produces 2% more wirelength and number of vias, 2.1% more non-preferred usage, 43.0% fewer shorts and 14.2% fewer violations. The ratio of short is relatively small due to a biased result on $ispd18\_test10\_metal5$ where our short is only 1.9% of CUGR (highlighted in red, 98.1% better). Excluding this benchmark, we still produce 21% fewer shorts and 3% fewer DRCs on the remaining 11 benchmarks.

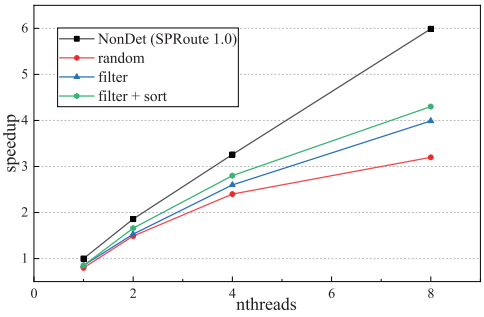


Fig. 3. Scalability of Maze Routing

To show the effectiveness of RUDY in soft capacity, we note that the quality score without RUDY in Table II drops by 0.6%. The decomposition of no-RUDY score is (not shown in the table), 102.0% wirelength, 119.9% non-preferred usage, 64.9% shorts and 96.7% violations compared with CUGR. Almost all metrics are degraded.

B. Performance of Bulk Synchronous Parallelization

We use the non-deterministic parallel maze routing in SPRoute 1.0 [2] as the baseline to study the parallel scalability. All the tuning parameters are the same and the only difference between the non-deterministic maze routing and SPRoute 2.0 is the bulk synchronous parallelization. The net usages in the non-deterministic maze routing are updated to global usage immediately without considering the conflict with other nets.

We study the performance on five benchmarks with $\_metal5$ in Fig. 3, as they are hard benchmarks and rip-up and reroute takes a substantial amount of time in the total runtime. $NonDet$ is the non-deterministic baseline SPRoute 1.0. $random$ represents a random scheduler which is line 11 in Algorithm 1. $filter$ means adding lines 2-5 in Algorithm 1 to filter out small nets in the first iteration. $filter + sort$ is our complete scheduler. On 8 threads, the $filter$ technique improves performance from 3.2X to 4.0X and $sort$ further improves to 4.3X.

We note that the difference between the quality scores of deterministic and non-deterministic maze routing is negligible ($< 0.1\%$).

VI. Conclusion

This paper is a contribution to detailed routability and deterministic parallelization in global routing. First, we introduce a soft capacity to reserve routing space for the detailed router based on the pin density and RUDY of the region. Second, we propose a bulk synchronous execution model for rip-up and reroute to achieve deterministic parallelization, in which the concurrent nets are not required to be disjoint. We design a scheduler to mitigate the load imbalance and livelock in the parallelization. We implement the proposed methodology in SPRoute 2.0 which generates good quality of results with less shorts and DRCs and a significant speedup over the state-of-the-art.

References

[1] 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.

TABLE II
COMPARISON WITH STATE-OF-THE-ART GLOBAL ROUTER

benchmarks	Quality Score			1 thread Runtime (s)		8 threads Runtime (s)	
	CUGR	ours	ours (no RUDY)	CUGR	ours	CUGR	ours
ispd18_test5	**18,125,476**	18,328,493	18,518,745	86.4	11.1	67.8	8.0
ispd18_test8	**42,922,183**	43,310,981	43,821,153	358.6	30.6	224.2	21.9
ispd18_test10	**45,740,937**	47,824,326	47,792,957	397.0	69.1	283.1	27.2
ispd18_test5_metal5	18,207,592	**18,156,169**	18,347,975	729.8	13.7	68.5	8.9
ispd18_test8_metal5	**42,093,805**	44,548,332	44,312,839	625.0	74.7	240.1	37.2
ispd18_test10_metal5	55,629,665	**52,510,117**	52,545,482	913.7	340.7	287.9	80.3
ispd19_test7	88,679,116	**87,357,811**	86,983,492	729.8	56.2	499.7	38.8
ispd19_test8	128,209,191	**127,619,707**	129,048,130	625.0	79.6	477.4	54.1
ispd19_test9	201,309,592	**199,900,533**	202,746,909	913.7	123.5	774.1	82.4
ispd19_test7_metal5	82,429,250	**81,209,006**	80,997,261	576.6	93.6	282.7	48.5
ispd19_test8_metal5	126,352,615	**126,349,648**	127,775,993	1206.9	305.3	449	146.5
ispd19_test9_metal5	197,959,113	**196,991,385**	199,723,710	1632.6	204.7	523.8	120.7
Ratio	100.0%	100.1%	100.7%	100.0%	14.5%	100.0%	13.5%
Speedup	-	-			6.9X		7.4X

TABLE III
DECOMPOSITION OF QUALITY SCORE

benchmark	Wire Length & Vias		Non-preferred Usage		Short		Min-area & Spacing	
	CUGR	ours	CUGR	ours	CUGR	ours	CUGR	ours
ispd18_test5	**17,472,005**	17,556,365	**149,496**	247,275	**234,973**	262,351	269,000	**262,500**
ispd18_test8	**42,131,417**	42,184,439	**260,204**	609,554	420,060	**416,487**	110,500	**100,500**
ispd18_test10	**44,042,368**	46,282,462	**890,438**	1,097,182	379,630	**76,680**	428,500	**36,680**
ispd18_test5_metal5	17,639,135	**17,462,085**	**126,936**	248,317	226,019	**224,764**	**215,500**	219,000
ispd18_test8_metal5	**41,247,938**	43,310,435	**335,782**	718,736	**424,584**	439,160	85,500	**80,000**
ispd18_test10_metal5	**45,088,773**	51,074,990	1,427,981	**1,134,460**	8,465,410	**157,665**	647,500	**143,000**
ispd19_test7	77,296,156	**77,070,664**	1,435,185	**754,712**	2,945,772	**1,704,932**	**7,002,000**	7,827,500
ispd19_test8	119,209,566	**118,949,420**	1,344,018	**980,890**	**1,668,505**	1,742,395	5,987,000	**594,700**
ispd19_test9	184,217,137	**183,856,190**	2,184,205	**1,635,075**	**3,884,747**	4,035,267	11,023,500	**10,374,000**
ispd19_test7_metal5	70,852,698	**71,888,122**	1,523,871	**796,540**	3,207,179	**1,983,842**	6,845,500	**6,540,500**
ispd19_test8_metal5	**116,106,260**	117,124,497	1,468,295	**1,053,695**	2,575,557	**2,575,557**	6,202,500	**6,133,000**
ispd19_test9_metal5	**179,247,362**	179,527,304	2,334,987	**1,742,730**	5,245,262	**4638350**	11,131,500	**11,083,000**
Ratio	100.0%	102.0%	100.0%	102.1%	100.0%	57.0%	100.0%	85.8%

[2] J. He, M. Burtscher, R. Manohar, and K. Pingali, "Sproute: A scalable parallel negotiation-based global router," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2019, pp. 1–8.

[3] S. Dolgov, A. Volkov, L. Wang, and B. Xu, "2019 cad contest: Lef/def based global routing," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2019, pp. 1–4.

[4] G. Chen, C.-W. Pui, H. Li, J. Chen, B. Jiang, and E. F. Young, "Detailed routing by sparse grid graph and minimum-area-captured path search," in *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, 2019, pp. 754–760.

[5] "Cadence innovus implementation system," http://www.cadence.com.

[6] P. Spindler and F. M. Johannes, "Fast and accurate routing demand estimation for efficient routability-driven placement," in *2007 Design, Automation & Test in Europe Conference & Exhibition*. IEEE, 2007, pp. 1–6.

[7] Y. Xu, Y. Zhang, and C. Chu, "Fastroute 4.0: Global router with efficient via minimization," in *2009 Asia and South Pacific Design Automation Conference*. IEEE, 2009, pp. 576–581.

[8] Y.-J. Chang, Y.-T. Lee, and T.-C. Wang, "Nthu-route 2.0: A fast and stable global router," in *2008 IEEE/ACM International Conference on Computer-Aided Design*. IEEE, 2008, pp. 338–343.

[9] H.-Y. Chen, C.-H. Hsu, and Y.-W. Chang, "High-performance global routing with fast overflow reduction," in *2009 Asia and South Pacific Design Automation Conference*. IEEE, 2009, pp. 582–587.

[10] Z. Qi, Y. Cai, Q. Zhou, Z. Li, and M. Chen, "Vfgr: A very fast parallel global router with accurate congestion modeling," in *2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2014, pp. 525–530.

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

[12] T. Taghavi, Z. Li, C. Alpert, G.-J. Nam, A. Huber, and S. Ramji, "New placement prediction and mitigation techniques for local routing congestion," in *2010 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2010, pp. 621–624.

[13] Y. Wei, C. Sze, N. Viswanathan, Z. Li, C. J. Alpert, L. Reddy, A. D. Huber, G. E. Tellez, D. Keller, and S. S. Sapatnekar, "Glare: Global and local wiring aware routability evaluation," in *DAC Design Automation Conference 2012*. IEEE, 2012, pp. 768–773.

[14] T.-H. Wu, A. Davoodi, and J. T. Linderoth, "A parallel integer programming approach to global routing," in *Design Automation Conference*. IEEE, 2010, pp. 194–199.

[15] ——, "Grip: Scalable 3d global routing using integer programming," in *Proceedings of the 46th Annual Design Automation Conference*, 2009, pp. 320–325.

[16] Y. Han, D. M. Ancajas, K. Chakraborty, and S. Roy, "Exploring high-throughput computing paradigm for global routing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 1, pp. 155–167, 2013.

[17] C. Chu and Y.-C. Wong, "Flute: Fast lookup table based rectilinear steiner minimal tree algorithm for vlsi design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 1, pp. 70–83, 2007.

[18] K. Pingali, D. Nguyen, M. Kulkarni, M. Burtscher, M. A. Hassaan, R. Kaleem, T.-H. Lee, A. Lenharth, R. Manevich, M. Méndez-Lojo *et al.*, "The tao of parallelism in algorithms," in *Proceedings of the 32nd ACM SIGPLAN conference on Programming language design and implementation*, 2011, pp. 12–25.