

LockRoute: A Spatial Locking Framework for Parallel Global Routing

Amber Thrall* Vidya Chhabria[†] S M Ferdous[‡] Mahantesh Halappanavar[§] Bala Krishnamoorthy*

*Washington State University, Vancouver, WA

{amber.thrall,kbala}@wsu.edu

[†]Arizona State University, Tempe, AZ

vidyachhabria@asu.edu

[‡]UNC Charlotte, Charlotte, NC

sferdou1@charlotte.edu

[§]Pacific Northwest National Laboratory, Richland, WA

hala@pnnl.gov

Abstract—Global routing is a crucial step in VLSI design that has become increasingly more complex as chip sizes and design scales grow. Many global routers divide the process into several stages: two-pin decomposition, congestion map generation, maze routing and layer assignment. Each of these stages requires routing thousands to millions of nets providing opportunities for parallelization. In this paper, we demonstrate that global routing is not embarrassingly parallel and describe a lock-based approach to shared-memory parallel global routing by identifying net dependencies through the use of an R-tree. This lock framework is flexible enough to be applied to every stage of the global routing pipeline. We evaluate our global router on the ISPD 2008 and ISPD 2019 global routing contest benchmark suites. Our approach achieves $3.45\times$ – $11.35\times$ speedup in runtime compared to other routers without a reduction in quality.

Index Terms—Parallel algorithms, Global routing, VLSI physical design, Multithreading, Spatial data structures, Synchronization

I. INTRODUCTION

As semiconductor technology scales and chip complexity grows, determining how to connect millions of circuit elements efficiently has become a major challenge in very-large-scale integration (VLSI) physical design. A critical stage of this process is global routing, in which the objective is to construct a coarse plan of interconnects that will carry signals between components on a chip. This coarse plan is used to guide a more detailed routing, in which the final placement of interconnects is set. Because the results of the global router directly influence the performance of detailed routing, improvements at this stage result in significant downstream impact on the overall design.

There has been considerable progress in accelerating global routing algorithms. For example, FastRoute [1] achieves high performance by exploring congestion-driven rectilinear Steiner minimum trees to reduce the need for expensive maze routing. Many global routers incorporate basic pattern routing techniques, such as L-pattern and Z-pattern, to drastically reduce runtime.

Beyond these algorithmic advances, parallel computing on modern multicore architectures offers additional, largely untapped potential for speedup. The most common approach to

parallel global routing is to split nets into batches, where each batch consists of non-overlapping nets that can be routed in parallel. Batch generation is a known bottleneck of parallel global routing, as the naive pairwise overlap comparisons between n nets require $O(n^2)$ time. Consequently, there have been many variations on this net-batching approach with recent approaches leveraging GPUs to accelerate net batching and/or the routing stages that follow, e.g., FastGR [2] and InstantGR [3].

An alternative approach to parallel computing was presented by NCTU-GR 2.0 [4], which employs locks to ensure that two-pin nets belonging to the same multi-pin net are routed sequentially. By preventing concurrent modifications to a single net’s routing resources, this approach mitigates race conditions associated with parallel routing. However, their method does not address race conditions that arise when two-pin nets from different multi-pin nets are routed in close proximity.

In this paper, we depart from the batching paradigm entirely and develop LockRoute, a global router that introduces a lock-based parallelization strategy built on spatial indexing structures, applied across every routing stage. Our strategy avoids the $O(n^2)$ overlap-comparison of net-batching and achieves significant speedup without degrading routing quality.

Our contributions are as follows:

- 1) A parallelization strategy that replaces net batching with spatially-indexed locking.
- 2) A spatially aware locking mechanism based on an R-tree structure that resolves conflicts between multi-pin nets.
- 3) Comprehensive empirical validation across two independent benchmark suites demonstrating consistent speedups with minimal degradation to routing quality.

To the best of our knowledge, this is the first global routing algorithm to employ spatially distributed locking for conflict-aware parallel maze routing.

On the benchmarks from the 2008 global routing contest hosted by ISPD, LockRoute attains a speedup of $5.13\times$ on average compared to NCTU-GR 2.0 [4] with minimal degradation to quality. This performance speedup is further

demonstrated on ISPD’s 2019 detailed routing contest benchmarks, commonly repurposed for global routing evaluation, in which LockRoute had speedup of $11.35\times$ compared to FastRoute and $3.45\times$ speedup compared to CuGR [5].

The remainder of this paper is laid out in four sections. In Section II, we overview the strategies employed by state-of-the-art global routers. In Section III, we present the details of LockRoute and its parallelization strategy. Finally, we compare the performance against three state-of-the-art global routers in Section IV and conclude in Section V.

II. PRELIMINARIES

We begin with a brief introduction to the various stages of a typical global router. For more details, we direct the reader to [6].

A. Outline of a Global Router

The goal of a global router is to provide a coarse routing of nets to guide future detailed routings. The majority of global routers make use of the grid graph model in which the chip area is partitioned into rectangular regions called global cells (G-cells) with all pins in a G-cell assumed to be in the center. Each G-cell corresponds to a node in the grid graph and two adjacent G-cells are connected together by an edge (see Figure 1).

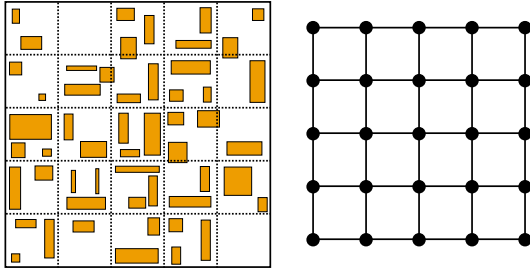


Fig. 1. Global routers typically partition the chip into global cells (left) and work on the corresponding grid graph (right).

A commonly used optimization objective is to minimize the total overflow on all edges of the grid graph. Each edge e is assigned a capacity c_e measuring the maximum number of nets that can traverse along that edge. Overflow occurs when an edge’s usage u_e exceeds the capacity, i.e., overflow of an edge e is defined by

$$o_e = \begin{cases} u_e - c_e & \text{if } u_e > c_e \\ 0 & \text{otherwise.} \end{cases}$$

A typical global router consists of several techniques designed to minimize the time spent performing costly maze routing. One commonly used approach is pattern routing, in which nets are routed by searching through predefined patterns, most often L-shaped or Z-shaped paths. Before such techniques can be applied, any multi-pin net needs to be decomposed into a collection of two-pin nets. This decomposition is typically performed by constructing a rectilinear minimum spanning tree (RMST) or by solving the more general rectilinear Steiner minimum tree (RSMT) problem.

B. The Steiner Tree Problem

Given a graph $G = (V, E)$ and a set of terminals $T \subseteq V$, the Steiner tree problem is to determine a set of Steiner points $S \subseteq V$ such that the tree spanning $T \cup S$ has minimal length.

State-of-the-art global routers often make use of FLUTE [7], an $O(n \log n)$ RSMT approximation algorithm (n is the number of terminals). It makes use of a lookup table for generating optimal RSMTs for nets of degree at most 9 and is still accurate for higher degree nets. FLUTE makes a good candidate for net decomposition as the majority of nets have low degrees in a typical problem. Moreover, FLUTE often provides smaller trees than rectilinear minimum spanning tree for any degree net [7].

C. L-Pattern and Z-Pattern Routing

The routing phase rips-up and reroutes all two-pin nets that are currently routed through a congested edge. Before getting into the slower maze routing algorithm, routers reduce the number of nets that need rerouting by performing L-pattern and Z-pattern routing.

L-pattern routing is a commonly used technique that attempts to connect a two-pin net using a simple, orthogonal path comprised of at most two rectilinear segments. Due to the small search space, L-pattern routing is significantly faster than maze routing and is often used to get an initial snapshot of the congestion. However, L-shaped paths cannot resolve cases where routing resources are blocked, and therefore require additional strategies such as Z-pattern routing and maze routing when congestion is high.

A Z-shaped pattern, also called a 2-bend pattern, is a rectilinear route connecting two-pins that has at most two bends in it. Z-pattern routing is more flexible than L-pattern routing while still being reasonably fast to perform. L-pattern and Z-pattern routing are typically performed in tandem.

D. Layer Assignment

The objective of layer assignment is to convert a 2D grid graph solution into one that makes use of the metal layers. For each two-pin net, we partition the net’s current route into segments which are then run through the layer assignment algorithm in sequence. The order in which segments are assigned can impact the solution. FastRoute 4.0 employs a complex ordering in which segments are ordered by their distances to their closest pin.

For a given segment $s = v_1^1 v_2^1 \dots v_k^1$, an L -by- k grid graph is constructed, where L is the number of layers. The objective is to find a shortest path from the source S node to the target T node. Without loss of generality, assume that S is on the left end of the grid and T is on the right end. If the target node is not known, treat each node on the right end as a potential target. The source node corresponds to v_1 and sits on the layer in which the previous segment terminated, or in the case of the first segment, the layer in which the pin lies.

Each node in the grid is assigned a cost $C(l, x)$ representing the number of vias required to travel from the source node to node v_x^l while avoiding congested edges. Once the costs have

been determined, the new route for the segment is extracted by back tracking from the target node to the source. Algorithm 1 describes how the costs are determined.

Algorithm 1 Layer Assignment of a Single Segment [1]

```

1: Initialize the cost  $C(l, x) = \infty$  for each node
2: For source  $S$  on layer  $l$ , set  $C(l, 1) = 0$ 
3: Update the costs for remaining layers in first column
4: for  $x = 2$  to  $k$  do
5:   for  $l = 1$  to  $L$  do
6:     if  $\text{cap}(\{v_{x-1}^l, v_x^l\}) > \text{usg}(\{v_{x-1}^l, v_x^l\})$  then
7:        $C(l, x) = C(l, x - 1)$ 
8:     end if
9:   end for
10:  Update the cost  $C(\cdot, x)$  for each layer
11: end for
12: Find the least cost for any target node and trace back

```

E. Maze Routing

At this stage, global routers begin ripping up and rerouting congested nets using maze routing. Many global routers employ shortest path algorithms such as Dijkstra’s algorithm to find the shortest path connecting the two-pin net while avoiding congested regions. Such approaches are dependent on the choice of a cost function and much discussion has revolved around the optimal cost function. Due to the computational cost of shortest path algorithms, maze routing is the slowest subroutine of global routing.

III. LOCKROUTE: A NEW GLOBAL ROUTER

We construct a simple, but effective, global router inspired by FastRoute 1.0 [8] and FastRoute 4.0 [1] that includes L/Z-pattern preprocessing phases.

A. Overflow Reduction Techniques

In the first phase, we use FLUTE [7] to construct a Steiner tree that is used to decompose the nets. Given a Steiner tree for a net, the edges of the tree are used to decompose the net into a set of two-pin nets. We then use the two-pin nets to get an initial congestion map in the following manner. For each two-pin net, we walk along the two L-shaped routes connecting the two pins and assign a usage of 0.5 along each edge in the route. If the two pins share an x or y coordinate, then the L-shaped routes are degenerate straight lines. In such an instance, we assign a usage of 1.0 along each edge. After assigning usage for each two-pin net, we perform a quick round of L-pattern routing. That is, for each two-pin net, we rip-up its usage along its two L-shaped routes and route the net along the L-shaped route with minimal usage. After performing L-shaped routing, we have an initial solution whose congestion is used to drive future rerouting.

In the next stage, we search through our currently routed two-pin nets given by L-pattern routing. For each two-pin net whose route traverse through a congested edge, we rip-up the current route and reroute using the Z-pattern route with

minimal usage. The combination of L-pattern routing followed by Z-pattern routing reduces the problem by over 99% in the hard-to-route benchmark instance newblue3 and completely solves some other instances.

Unlike other global routers, our approach converts the 2D solution given in phases one and two into a 3D solution by assigning routed nets to specific metal layers prior to maze routing. By performing layer assignment before maze routing, we significantly reduce the number of nets that need rerouting, thereby lowering the number of maze routing iterations needed to achieve low overflow. Since we perform layer assignment at this stage, each subnet consists of at most three segments. This structure allows us to streamline the routing strategy used by other global routers, such as FastRoute 4.0, by ordering segments using a straightforward graph traversal rather than relying on more complex orderings.

B. 3D Maze Routing

For edges within a layer, we make use of the same cost function as FastRoute 1.0 [8] shown in equation (1) that makes use of a logistic function with parameters h and k :

$$\text{cost}(e) = 1 + \frac{h}{1 + \exp(-k(u_e - c_e))}. \quad (1)$$

The goal of the cost function is to rapidly increase as the usage u_e nears or exceeds the capacity c_e , but increases slowly in under-capacity or over-capacity regions. The authors of FastRoute do not specify a choice of parameters. Through experimentation, we found that $k = 1$ and $h = \min\{c_e \mid e \in E\}$ perform well.

For via edges, we simply apply a constant cost β . The choice for β gives a trade-off between total overflow and number of vias, where small values of β result in an increased number of vias but smaller total overflow. By default, we set $\beta = 1$.

By default, our global router repeatedly applies maze routing until the total overflow stops decreasing.

C. Parallelization

A typical problem instance has upwards of a million nets that need to be solved. Global routers typically process nets sequentially, which leads to a long runtime and poor scalability. This naturally motivates the need for effective parallelization strategies within the global routing pipeline. However, as our experimental results demonstrate (Section IV, Table I), naively parallelizing routing can degrade solution quality, demonstrating that parallelism is itself a design challenge. In this section, we describe our approach to introduce multithreading into the global routing pipeline.

A common strategy in parallel computing is to make use of locks, which provide a synchronization mechanism to ensure safe concurrent access to shared data. The idea of a lock is to restrict access to certain regions of memory so that only one thread can modify that region at a given time, thereby preventing data races and inconsistencies in shared state. We make use of locks that block regions of the grid graph by distributing locks with an R-tree.

R-trees, introduced by Guttman in 1984 [9], are tree data structures for spatial indexing. Each node of the tree, called pages, are comprised of M entries. Each entry is given a bounding box in which the data of said entry are contained. For our purposes, the entries for leaf pages correspond to nets. In a non-leaf node, each entry points to a child node whose geometry is contained within the entry's bounding box. Figure 2 gives an example R-tree construction where $M = 3$.

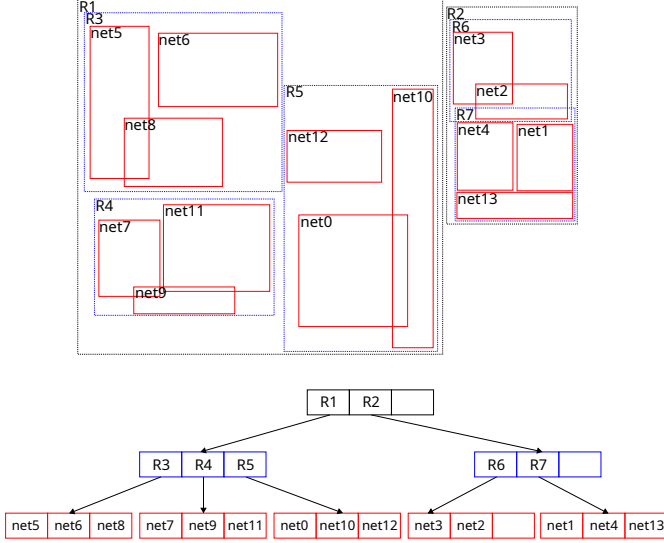


Fig. 2. Sample R-tree for a 13 net design with $M = 3$. Each leaf node (R3, R4, R5, R6, R7) is assigned a lock.

By using an R-tree, nets that are spatially clustered are assigned to the same leaf node. While leaves are not strictly guaranteed to be disjoint (see leaf nodes R6 and R7 in Figure 2) since the tree is bounded by the entry limit M and may subdivide space in a way that causes bounding boxes to overlap, our construction prioritizes the grouping of nets by locality. As a result, it is preferential to route nets within the same leaf sequentially. For instance, net7, net9, and net11 in Figure 2 are spatially clustered; routing them in parallel would increase the likelihood of conflict, hence they should be routed sequentially. Nets contained in different R-tree leaves are safer to route in parallel, since greater spatial separation reduces the probability that the corresponding threads will attempt to modify the same grid cells.

Our strategy (Algorithm 2) constructs an R-tree and assigns a lock to each leaf node. For instance, the example in Figure 2 will have five locks. Before routing a specific net N , we quickly query the R-tree to determine which leaf net N belongs to and set the lock corresponding to said leaf. If the lock is already set, the thread sits and waits for the lock to become available. In the case of a two-pin net, we query the R-tree for the net's parent net. This strategy is applied to every routing stage of the global routing pipeline.

The choice for the maximum number of entries per node, M , introduces a hyperparameter. Smaller values of M result in faster runtime but higher probability of conflicting nets being

Algorithm 2 Parallel Router Using Locks

- 1: Sort nets $\mathcal{N}$ by perimeter
 - 2: $T \leftarrow \text{BUILDRTREE}(\mathcal{N})$
 - 3: **parfor all** Nets N **do**
 - 4: $L \leftarrow \text{FINDLEAF}(T, N)$
 - 5: Set lock L
 - 6: If current route is congested, rip-up and reroute.
 - 7: Unset lock L
 - 8: **end parfor all**
-

routed in parallel. Whereas larger values of M result in slower runtime but less conflict. Through experimentation, we found that setting $M = 32$ obtained the best results.

There are several bulk construction methods for R-trees. We employ the Sort-Tile Recursive (STR) method introduced by Leutenegger et al. [10]. STR constructs an R-tree by partitioning and ordering nodes based on their position in space. More specifically, the nodes are first sorted by their x -coordinates and then partitioned into at most $\sqrt{\lceil n/M \rceil}$ partitions, each of which are then sorted by the y -coordinates. Every M nodes are then merged based on the resulting ordering. This process is repeated until a single root node is obtained. See Algorithm 3 for more details.

Algorithm 3 Sort-Tile-Recursive [10]

- 1: Construct a node for each net.
 - 2: **while** Number of parent nodes is greater than 1 **do**
 - 3: Sort all nodes by x -coordinate of the center of bounding boxes.
 - 4: Divide all n nodes into $\sqrt{\lceil n/M \rceil}$ partitions, each partition is sorted by y -coordinate of the center of bounding boxes.
 - 5: Merge every M nodes into a new parent node.
 - 6: **end while**
-

One benefit of the R-tree over other hierarchical spatial indexes is a favorable computational complexity. Under the typical low overlap scenario, an R-tree supports queries in $O(\log_M n)$ time, with a worst-case complexity of $O(n)$ [11]. Construction is also efficient; for example, the STR bulk-loading algorithm runs in $O(n \log n)$ time [12]. In contrast to the computational cost of maze routing, our strategy incurs minimal overhead.

Our R-tree construction imposes a cap, M , on the number of nets assigned to a given leaf node; overlapping nets that exceed this cap are routed without conflict resolution. Across the ISPD'08 benchmarks, this resulted in an average of $c_i/N_i \approx 0.012$ unresolved potential conflicts per net routed; however, this number varied across benchmarks (0.002–0.110). We did not observe a measurable degradation in routing quality attributable to these unresolved conflicts, although we flag this as a current limitation. In future work, we aim to address this shortcoming using dynamic leaf capacity.

To further reduce conflicts, all fetches and updates to a grid edge's usage are performed using atomic operations.

IV. EXPERIMENTAL RESULTS

Our framework, named LockRoute, was implemented¹ in C++ using OpenMP [13] and tested on ISPD’08 global routing contest benchmarks [14]. We first demonstrate the scalability of LockRoute by comparing runtime against number of threads. We then compare LockRoute to three other global routers: NCTU-GR 2.0 [4], FastRoute [1], and CuGR [5]. NCTU-GR 2.0 was accessed through the authors web page, whereas FastRoute and CuGR were accessed through OpenROAD [15]. Due to file incompatibility with ISPD’08 benchmarks, FastRoute and CuGR were tested on ISPD’19 detailed routing contest benchmarks [16].

A. Scalability Results

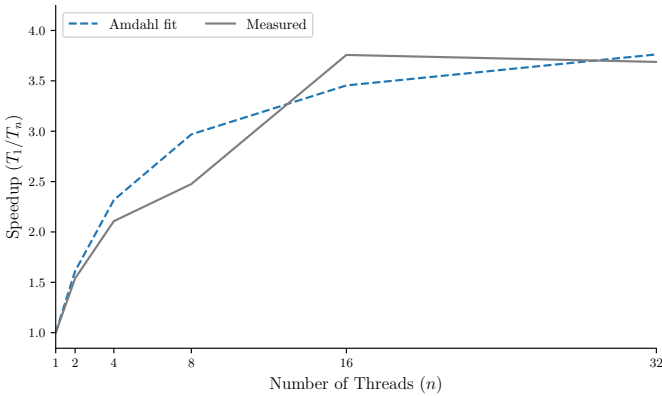


Fig. 3. Scalability of LockRoute showing time speedup factor ($y \times$) with increasing number of threads.

LockRoute’s scalability was evaluated against ISPD’08 benchmark newblue3 on an AMD EPYC 7502 32-core processor with 256 GB of DDR-3200 memory. The benchmark newblue3 was selected because it is the most computationally demanding instance in the ISPD’08 suite. Each experiment was repeated 10 times at varying thread counts, and the average runtime was recorded.

In addition to runtime, we measured total overflow (TO) and max overflow (MO) to verify that the solution quality was maintained. As shown in Figure 3, LockRoute achieves substantial speedup as the number of threads increases. Furthermore, the total overflow and max overflow were consistent despite the change in thread count (see Table I).

Table I also demonstrates the necessity of a parallelization strategy as all benchmark instances exhibit increased overflow when solved without synchronization. Through the use of locks, we get significant speedup while avoiding the degradation in quality that occurs without a parallelization strategy.

We used Amdahl’s law to estimate the theoretical speedup by fitting a function of the form $1/((1-p) + p/n)$ where p is the fraction of the workload that can be parallelized and n is the number of threads. The fitted model estimates an

effective parallel fraction of 75.8% ($p \approx 0.758$), resulting in a theoretical maximum speedup of $4.13 \times$ as $n \rightarrow \infty$.

TABLE I
OVERFLOW RESULTS FOR VARIOUS STRATEGIES. MO AND TO ARE MAX AND TOTAL OVERFLOWS. Locks AND No Locks STRATEGIES RAN USING 32 THREADS, WHEREAS Sequential RAN SINGLE-THREADED.

Benchmark	Sequential		Locks		No Locks	
	MO	TO	MO	TO	MO	TO
adaptec1	0	0	0	0	1	24
adaptec2	0	0	0	0	2	37
adaptec3	0	0	0	0	1	32
adaptec4	0	0	0	0	1	18
adaptec5	0	0	0	0	2	32
bigblue1	0	0	0	0	2	34
bigblue2	0	0	0	0	2	19
bigblue3	0	0	0	0	2	27
bigblue4	0	0	0	0	2	16
newblue1	0	0	0	0	2	10
newblue2	0	0	0	0	1	19
newblue3	263	5705	235	5833	189	10426
newblue4	0	0	0	0	2	39
newblue5	0	0	0	0	2	28
newblue6	0	0	0	0	2	32
newblue7	0	0	0	0	2	60

B. ISPD’08 Benchmarks

We compared LockRoute against NCTU-GR 2.0 on the ISPD’08 benchmarks. Both global routers were run on an AMD Ryzen 9 9900X 12-Core Processor with 64 GB of DDR5 memory. The ISPD’08 benchmarks were split into two groups, overflow-free (shown in Table II) and hard-to-route (shown in Table III), based on NCTU-GR 2.0 results. To save space in the table, wirelength (WL) and number of vias were divided by 10^5 .

TABLE II
QUALITY AND RUNTIME RESULTS COMPARING LOCKROUTE AGAINST NCTU-GR 2.0 ON OVERFLOW-FREE CASES. WL AND # VIA GIVE THE WIRELENGTH AND THE NUMBER OF VIAS IN UNITS OF 10^5 . CPU GIVES RUNTIME IN SECONDS.

Benchmark	LockRoute			NCTU-GR 2.0		
	WL	# VIA	CPU (s)	WL	# VIA	CPU (s)
adaptec1	33.91	27.14	2.98	37.83	20.38	11.54
adaptec2	32.13	27.01	1.71	34.04	20.04	7.82
adaptec3	93.56	50.33	4.49	97.78	38.49	19.12
adaptec4	89.02	44.63	4.05	90.28	38.82	16.12
adaptec5	98.12	77.19	5.07	107.87	58.33	40.55
bigblue1	34.34	29.67	7.01	41.57	23.43	23.69
bigblue2	46.32	52.87	3.68	49.93	45.04	27.34
bigblue3	77.06	70.97	5.54	80.44	53.28	20.19
newblue1	23.41	30.22	1.80	25.03	23.08	23.97
newblue2	46.17	37.79	2.40	46.83	29.80	9.94
newblue5	142.52	126.74	10.79	152.75	89.58	55.43
newblue6	97.83	106.54	7.01	110.48	82.77	37.21

Speedup was measured using the per-benchmark ratios $s_i = T_{\text{NCTU-GR},i} / T_{\text{LockRoute},i}$, where $T_{\text{NCTU-GR},i}$ and $T_{\text{LockRoute},i}$ denote the NCTU-GR and LockRoute runtime measurements for benchmark i , respectively. Speedup is reported as the geometric mean of ratios $GM_{\text{CPU}} = \exp(\frac{1}{n} \sum_i \ln s_i)$. For

¹Available at <https://github.com/AmberThrall/LockRoute>

TABLE III
QUALITY AND RUNTIME RESULTS COMPARING LOCKROUTE (12 THREADS) AGAINST NCTU-GR 2.0 ON HARD-TO-ROUTE CASES. COLUMN LABELS ARE SAME AS THOSE IN TABLES I AND II.

Benchmark			LockRoute			NCTU-GR 2.0				
	MO	TO	WL	# VIA	CPU (s)	MO	TO	WL	# VIA	CPU (s)
bigblue4	0	0	116.25	145.50	12.24	8	1056	127.06	113.42	50.39
newblue3	246	5868	74.28	36.35	225.04	788	36850	38.03	80.04	80.16
newblue4	0	0	79.66	63.92	3.50	2	216	84.91	49.52	31.64
newblue7	0	0	178.06	209.99	12.39	2	792	191.30	170.80	168.54

TABLE IV
QUALITY AND RUNTIME RESULTS COMPARING LOCKROUTE (12 THREADS) AGAINST FASTROUTE AND CuGR. COLUMN LABELS ARE SAME AS THOSE IN TABLES I AND II. MAXIMUM OVERFLOW VALUES ARE OMITTED FOR CuGR WHEN THEY CANNOT BE RELIABLY DETERMINED, SINCE CuGR DOES NOT REPORT THIS METRIC.

Benchmark			LockRoute			FastRoute					CuGR				
	MO	TO	WL	# VIA	CPU (s)	MO	TO	WL	# VIA	CPU (s)	MO	TO	WL	# VIA	CPU (s)
ispd19_test1	0	0	0.61	0.20	0.04	0	0	0.87	0.40	0.53	0	0	0.85	0.39	0.29
ispd19_test2	0	0	24.24	3.78	0.59	0	0	30.08	8.66	15.34	0	0	29.48	8.09	3.49
ispd19_test3	0	0	0.80	0.26	0.03	0	0	1.21	0.71	0.41	-	6	1.16	0.65	0.36
ispd19_test4	0	0	51.27	2.40	0.91	0	0	62.79	3.34	15.23	-	116	69.94	7.88	5.64
ispd19_test6	1	23	64.49	10.41	10.32	0	0	79.36	21.66	43.76	0	0	77.90	20.50	9.40
ispd19_test7	0	0	118.46	18.72	5.47	0	0	146.73	42.18	79.09	0	0	144.53	38.29	19.54
ispd19_test8	1	8	183.01	30.96	25.57	0	0	226.42	64.71	130.17	0	0	222.01	63.07	27.02
ispd19_test9	1	1	276.40	51.31	34.50	0	0	348.12	107.10	194.00	0	0	340.53	105.01	46.18
ispd19_test10	0	0	272.24	46.07	9.59	0	0	344.81	106.18	217.02	0	0	335.86	96.62	45.81

wirelength, improvement is reported as the aggregated percentage reduction given by $(1 - GM_{WL}) \times 100\%$ where $GM_{WL} = \exp\left(\frac{1}{n} \sum_i \ln \frac{WL_{LockRoute,i}}{WL_{NCTU,i}}\right)$. A similar metric as wirelength is used for via count.

In our experiments, LockRoute had a $5.13\times$ speedup with 7.23% decrease in wirelength with only a 29.07% increase in number of vias on overflow-free instances. For the hard-to-route instances, there was a $3.66\times$ speedup with 11.76% increase in wirelength and 1.94% decrease in number of vias. Hence, LockRoute achieves a significant speedup compared to NCTU-GR while having minimal degradation to quality.

Of particular note is the improvement in overflow on the hard-to-route instances. LockRoute has a 84.08% reduction in total overflow on newblue3 and achieved zero overflow on bigblue4, newblue4, and newblue7. While LockRoute took significantly longer than NCTU-GR 2.0 on newblue3, this time is outweighed by the significant reduction in overflow. Further experiments showed that if one restricts newblue3 to a single iteration, LockRoute achieves a total overflow of 7586 (79.41% reduction) in only 38.93 seconds ($2.06\times$ speedup).

C. ISPD'19 Benchmarks

We also compare LockRoute against FastRoute and CuGR on the ISPD'19 benchmarks in Table IV. These benchmarks were run on the same machine as the ISPD'08 benchmarks.

We again computed the aggregate percent change comparing LockRoute to FastRoute and CuGR. Compared to FastRoute, LockRoute had a $11.35\times$ speedup with a 22.45% decrease in wirelength and 52.55% decrease in number of vias. When compared to CuGR, LockRoute had a $3.45\times$ speedup with

a 21.71% decrease in wirelength and 54.54% decrease in number of vias.

These speedup results are consistent with those observed on the ISPD'08 benchmarks. However, unlike the ISPD'08 benchmarks, LockRoute had a significant reduction in the number of vias while attaining less wirelength. This outcome is particularly desirable, as vias are well known to contribute to manufacturing defects.

V. CONCLUSION

We presented a new global router that focuses on reducing runtime through parallelization. Our R-Tree lock framework is highly flexible and was applied to every stage of global routing. LockRoute consistently outperformed existing academic global routers, achieving over $10\times$ faster runtime on average while maintaining comparable results.

Despite these improvements, the proposed parallelization strategy has notable limitations. As discussed in Subsection III-C, the R-tree leaves that define lock regions are not guaranteed to be disjoint. This reduces, but does not eliminate, the possibility of routing conflicts among threads. Further work should explore improved lock-distribution schemes and alternative spatial partitioning structures. In addition, our current maze router does not take advantage of the more modern techniques used by global routers.

Our LockRoute router demonstrates that a well-designed parallelization strategy can substantially reduce global routing time, and we view this work as a step toward fully exploiting parallel computing for large-scale physical design.

ACKNOWLEDGMENTS

This work was supported in part by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research's Advanced Computing Technologies Competitive Portfolios program at Pacific Northwest National Laboratory (PNNL). PNNL is a multi-program national laboratory operated for the U.S. Department of Energy (DOE) by Battelle Memorial Institute under Contract No. DE-AC05-76RL01830. Amber Thrall was supported in part by the Pacific Northwest National Laboratory–Washington State University Distinguished Graduate Research Program (PNNL-WSU DGRP).

REFERENCES

- [1] 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*, 2009, pp. 576–581.
- [2] 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.
- [3] S. Lin, L. Xiao, J. Liu, and E. F. Y. Young, "InstantGR: Scalable GPU Parallelization for Global Routing," in *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*, ser. ICCAD '24. New York, NY, USA: Association for Computing Machinery, 2025.
- [4] 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.
- [5] 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.
- [6] H. Tang, G. Liu, X. Chen, and N. Xiong, "A Survey on Steiner Tree Construction and Global Routing for VLSI Design," *IEEE Access*, vol. 8, pp. 68 593–68 622, 2020.
- [7] 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, 2008.
- [8] 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.
- [9] A. Guttman, "R-trees: A dynamic index structure for spatial searching," *SIGMOD Rec.*, vol. 14, no. 2, pp. 47–57, Jun. 1984.
- [10] S. T. Leutenegger, J. M. Edgington, and M. A. Lopez, "STR: A Simple and Efficient Algorithm for R-Tree Packing," Institute for Computer Applications in Science and Engineering, NASA Langley Research Center, Hampton, VA, Contractor Report NASA CR-202 321 / ICOMP-97-02 E-10638, February 1997, unclassified; unlimited, public domain. Accession ID: 97N19343. [Online]. Available: https://archive.org/details/nasa_techdoc_19970016975
- [11] P. Rigaux, M. O. Scholl, and A. Voisard, *Spatial Databases: With Application to GIS*, ser. Series in Data Management Systems. Elsevier Science, 2002. [Online]. Available: <https://books.google.com/books?id=o8LfhpFOnPwC>
- [12] I. Kamel and C. Faloutsos, "On packing R-trees," in *Proceedings of the Second International Conference on Information and Knowledge Management*, ser. CIKM '93. New York, NY, USA: Association for Computing Machinery, 1993, p. 490–499. [Online]. Available: <https://doi.org/10.1145/170088.170403>
- [13] L. Dagum and R. Menon, "OpenMP: An industry standard API for shared-memory programming," *IEEE Computational Science and Engineering*, vol. 5, no. 1, pp. 46–55, 1998.
- [14] G.-J. Nam, C. Sze, and M. Yildiz, "The ISPD global routing benchmark suite," in *Proceedings of the 2008 International Symposium on Physical Design*, ser. ISPD '08. New York, NY, USA: Association for Computing Machinery, 2008, pp. 156–159. [Online]. Available: <https://doi.org/10.1145/1353629.1353663>
- [15] T. Ajayi, D. Blaauw, T.-B. Chan, C.-K. Cheng, V. A. Chhabria, D. K. Choo, M. Coltella, R. G. Dreslinski, M. Fogaca, S. M. Hashemi, A. A. Ibrahim, A. B. Kahng, M. Kim, J. Li, Z. Liang, U. Mallappa, P. I. Péntzes, G. Pradipta, S. Reda, A. Rovinski, K. Samadi, S. S. Sapatnekar, L. K. Saul, C. Sechen, V. Srinivas, W. Swartz, D. Sylvester, D. Urquhart, L. Wang, M. Woo, and B. Xu, "OpenROAD: Toward a Self-Driving, Open-Source Digital Layout Implementation Tool Chain," in *Proceedings of the Government Microcircuit Applications and Critical Technology Conference (GOMACTech)*, 2019, <https://vlsicad.ucsd.edu/Publications/Conferences/370/c370.pdf>.
- [16] W.-H. Liu, S. Mantik, W.-K. Chow, Y. Ding, A. Farshidi, and G. Posser, "ISPD 2019 Initial Detailed Routing Contest and Benchmark with Advanced Routing Rules," in *Proceedings of the 2019 International Symposium on Physical Design*, ser. ISPD '19. New York, NY, USA: Association for Computing Machinery, 2019, pp. 147–151. [Online]. Available: <https://doi.org/10.1145/3299902.3311067>