A Runtime Optimization Approach for FPGA Routing

Dekui Wang, Zhenhua Duan, *Senior Member, IEEE,* Cong Tian, *Member, IEEE*, Bohu Huang, and Nan Zhang

***Abstract*—In this paper, we present a new field-programmable gate array (FPGA) routing approach on the basis of the PathFinder routing algorithm. During each routing iteration, our approach applies a novel timing-based rerouting strategy to only reroute the illegal paths. At a lower level, each maze expansion is started from the relatively close part of current routing tree to search for the target sink on the routing resource graph. Experimental results demonstrate that on average the proposed approach reduces the routing runtime by 68.5% compared with the timing-driven router in versatile place and route FPGA placement and routing framework, with reduction of 2.5% and 1.4% in critical path delay and wirelength, respectively.**

***Index Terms*—Field-programmable gate array (FPGA) routing, maze router, PathFinder, rerouting strategy, routing algorithm.**

I. Introduction

As field-programmable gate arrays (FPGAs) get larger and more complex, the associated FPGA computer-aided design (CAD) tools require more runtime to synthesize large FPGA designs. In addition, the worsening FPGA CAD tool runtime will continue to grow with the rapidly increasing device resources within FPGAs. Consequently, it is invaluable to reduce the compilation time, so that the time cost is decreased in the development of FPGA-based systems.

Routing is one of the time-consuming steps of the FPGA CAD flow. The existing work on reducing router runtime mainly focuses on parallelization [1]–[3] or tradeoff between runtime and the quality of results [1], [4], which are also dominated by PathFinder algorithm. In the original PathFinder, each iteration reroutes all the paths. However, some existing paths built in the previous iteration are not worse than the paths found in the current iteration. Further, in the maze routing step, the maze expansion starts from the entire routing tree. Nevertheless, many nodes on the routing tree are almost impossible to be on the lowest cost path when the routing tree spans a large area. Accordingly, it is likely to reduce runtime by improving the rerouting strategy and the maze expansion.

In this paper, we propose a new FPGA routing approach based on the PathFinder algorithm. With our approach, we set up two timing-based conditions called timing conditions. We refer to congested paths and the paths satisfying timing conditions as illegal paths, and the others as legal ones. Basically, the proposed approach iteratively routes each net by invoking an improved net router. With a new rerouting strategy, the improved net router only reroutes the illegal paths, leaving the legal ones unchanged. At the lowest level, we improve the maze router by initially expanding the wavefront out from part of the current routing tree. We implement and demonstrate our approach using the versatile place and route (VPR) framework [5].

The main contributions of this paper are as follows.

1) We improve the maze expansion step in the original PathFinder algorithm. When routing a sink, the improved maze router initially expands only a part of the current routing to route the target sink, rather than the entire routing tree as the original PathFinder does.
2) We present a new rerouting strategy for the net router. When rerouting a net, the improved net router only reroutes the illegal paths, keeping the legal ones unchanged.
3) We set up timing conditions to pick out the paths that are likely to harm the critical path delay. Experimental results show that on average the proposed approach not only achieves a 68.5% reduction in runtime, but also reduces critical path delay by 2.5% and wirelength by 1.4%.

The remainder of this paper is organized as follows. Section II provides a brief description of the proposed approach. Further, the improved maze router is presented in Section III and a new rerouting strategy is presented in Sections IV and V. In Section VI, the experimental results are demonstrated. Section VII reviews the related work for accelerating FPGA routing. Finally, the conclusions are drawn in Section VIII.

II. Runtime Optimization Routing Approach

This section describes our proposed FPGA routing approach. Although the approach also applies the idea of negotiated congestion routing, we significantly improve the maze router and the rerouting strategy of the net router in the original PathFinder negotiation congestion algorithm.

During the first iteration, our approach relies on an improved maze router to route each sink. This iteration is basically the same as in the original PathFinder algorithm. However, in subsequent iterations, the improved net router applies a new rerouting strategy and invokes the improved maze router to reroute the illegal paths. The process of rerouting the illegal paths continues until a feasible routing result is found or the maximum number of iterations is reached.

At the heart of our approach, the improved maze router routes one sink at a time and is particularly described in Section III. During the maze routing of a sink, the wavefront is initialized using the close part of the routing tree. As the initialization is complete, the lowest cost node $v_{\min}$ is removed from the wavefront. If $v_{\min}$ is target sink t, then the path from source to t is constructed, and the maze expansion terminates. Otherwise, the improved maze router adds the nodes adjacent to $v_{\min}$ to the wavefront and continues to explore the neighbors of the lowest cost node.

When rerouting a net, the improved net router applies a new rerouting strategy and first checks this net's routing tree to rip-up the illegal paths, and adds the corresponding sinks to a set s. After that, the improved maze router is invoked to route the sinks in s. In this way, the legal paths remain unchanged and only the illegal ones are rerouted, and the details are described in Sections IV and V.

Manuscript received December 28, 2016; revised June 2, 2017 and August 12, 2017; accepted October 25, 2017. Date of publication October 31, 2017; date of current version July 17, 2018. This work was supported by the NSFC under Grant 61420106004, Grant 61732013, Grant 61572386, and Grant 61672403. This paper was recommended by Associate Editor S. Reda. *(Corresponding authors: Zhenhua Duan; Cong Tian.)*

The authors are with the ICTT and ISN Laboratory, Xidian University, Xi'an 710071, China (e-mail: dekui_wang@126.com; zhhduan@mail.xidian.edu.cn; ctian@mail.xidian.edu.cn; huangbo@mail.xidian.edu.cn; nzhang@mail.xidian.edu.cn).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2017.2768416

At a higher level, the global router invokes the improved net router to route each net, resulting in an intermediate routing result. Subsequently, the proposed approach increases the costs of overused resources and makes nets with alternative routes avoid using these resources. If there is a overused resource, the routing result is illegal and the routing process continues; otherwise, it is legal and the routing process terminates.

III. Improved Maze Router

In this section, we improve the maze router in the original PathFinder routing algorithm. In general, the maze routing process can be divided into two steps: 1) initializing the maze routing wavefront and 2) maze expansion.

A. Maze Routing Wavefront Initialization

In the improved maze routing algorithm, a relatively small box $s-$box is applied for each pair of source and sink. At the beginning of the maze routing, the wavefront is initialized using only the part of current routing inside $s-$box. For each node v on the current routing tree, if v is inside the $s-$box of the target sink t, the existing path from source to v is partial route from source to t and the path cost from source to t is

$$Tcost(v) = Bcost(v) + \alpha \times e_{v,t} \tag{1}$$

where $Bcost(v)$ is the cost of the backward path from source to node v, and $e_{v,t}$ is the estimated cost of the path from v to t, and α is used to balance the effect of $e_{v,t}$. Typically, $Bcost(v)$ is defined as follows:

$$Bcost(v) = c_t \times \mathrm{del}_v \tag{2}$$

where c_t is the criticality of t and del_v is the delay from source to v. The wavefront initialization traverses all of the nodes on the current routing tree.

B. Maze Expansion

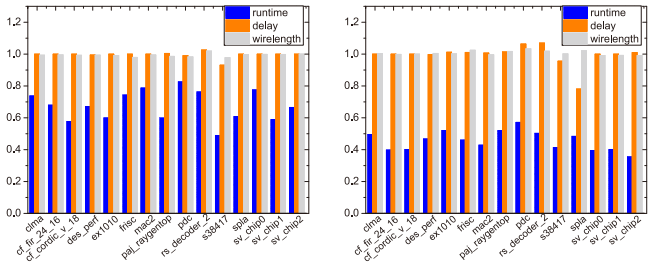
After the wavefront initialization, the improved maze router then expands the wavefront out to explore the nodes until the target sink is reached. During the maze expansion step, the improved maze router first removes the lowest cost node $v_{\min}$ from the wavefront and then expands $v_{\min}$ to visit its neighbors. For each neighbor v of $v_{\min}$, if v is the target sink, the maze expansion terminates and a backtrace procedure is called to construct the path from v to the source. Otherwise, the improved maze router applies a cost function at node v and adds node v to the wavefront. Here, if v belongs to this net's routing tree, the cost of the backward path $Bcost(v)$ is computed using (2) and the path cost from source to target sink t is computed using (1). Otherwise, $Bcost(v)$ is computed using

$$Bcost(v) = Bcost(v_{\min}) + \mathrm{Cost}(v) \tag{3}$$

where $Bcost(v_{\min})$ is the cost of the previous path from the source to $v_{\min}$, and $\mathrm{Cost}(v)$ is the incurred cost by using v

$$\mathrm{Cost}(v) = c_t \times d_v + (1 - c_t) \times b_v \times p_v \times h_v \tag{4}$$

where c_t is the criticality of t, d_v the intrinsic delay of v, b_v the base cost of using v, p_v the present congestion cost of v, and h_v the historical congestion of v. These parameters are usually constants determined by VPR. The maze expansion continues until t is reached. After t is found, the path from the source to t is the part of this net's routing tree and the costs of the routing resources on this path are updated.



(a) (b)

Fig. 1. Comparisons of runtime, critical path delay, and wirelength of baseline VPR to VPR enhanced by the (a) improved maze router and (b) VPR improved by skipping the rerouting of uncongested paths.

C. Setting the Size of $s-$Box

The $s-$boxes of each pair of source and sink are computed before the first routing iteration begins. If $s-$box is set too small, the improved maze router may expand the wavefront out for a large number of times before the target sink is found. On the contrary, if $s-$box is set too large, it takes much time to initialize the wavefront.

With our approach, $s-$box is set to n channels outside the bounding box of source and sink, and n is set to an appropriate value by experiments. As expected, experimental results show that the paths joining the routing trees at the nodes outside $s-$boxes is about 1% of all the paths when n equals 0; whereas, this figure is less than 0.1% when n is greater than 0, which is almost negligible. In order to balance the wavefront initialization and maze expansion, we accordingly set n to 1.

D. Effectiveness of the Improved Maze Router

Experiments are conducted to evaluate the improved maze router. Fig. 1(a) shows the ratios of runtime, critical path delay, and wirelength of VPR enhanced by the improved maze router to that conducted by baseline VPR. As we can see, the improved maze router reduces runtime by 33.1% over baseline VPR without negative effects on the quality of results. For most of the benchmark circuits, the quality of results remains unaffected or even improved. On average, the improved maze router reduces the critical path delay and wirelength by 0.4% and 0.7%, respectively.

IV. Skipping the Rerouting of Uncongested Paths

In this section, we propose a new rerouting strategy by which relatively fewer paths are rerouted in the rerouting of each net. We also present the benefits on runtime and analyze the effects of the proposed rerouting strategy on the quality of results.

A. Motivation

We refer to the paths using overused nodes as congested paths, and the others as uncongested ones. For a sink t, the lowest cost path from source to t found in kth iteration is denoted by P_k. Typically, the cost of a path P is

$$C(P) = \sum_{i=1}^{|P|} \mathrm{Cost}(n_i) \tag{5}$$

where $|P|$ denotes the length of P, n_i $(1 \le i \le |P|)$ is a node on P and $\mathrm{Cost}(n_i)$ is defined as in (4). Furthermore, we also conduct experiments to compare the costs of P_k and P_{k-1} in kth $(k \ge 2)$ iteration. If $C(P_{k-1}) \le C(P_k)$, we say that P_{k-1} is heritable. By the experiments, on average 74.6% of the uncongested paths are heritable while only 24.5% of the congested paths have heritability. Usually, the overused

nodes are imposed with high historical congestion costs. Therefore, when rerouting a congested path, it is likely for the maze router to find a path with lower cost. Motivated by experimental results, accordingly the coarse rerouting strategy in the original PathFinder algorithm is improved by only rerouting the congested paths in each iteration except the first one.

B. Rerouting the Congested Paths

Similar to the work presented in [1], we go a step further and present a new rerouting strategy to skip the rerouting of uncongested paths. During each iteration, only the congested paths are rerouted, and the uncongested ones remain unchanged. In this way, the proposed approach is free from the rerouting of uncongested paths.

When rerouting a net, with the proposed rerouting strategy, the improved net router first checks this net's routing tree RT built in the previous iteration to rip-up the congested paths. If a node on RT is overused, the paths using this node are actually congested and hence all the nodes on these congested paths are released. After all the congested paths on RT are ripped-up, the current RT contains only uncongested paths and then the timing analysis of RT is carried out. Subsequently, the proposed approach invokes the improved maze router to route the unconnected sinks. In general, there are relatively fewer congested paths in the later routing iterations and hence runtime of each iteration decreases as the iteration count increases. After all the unconnected sinks are routed, the current net's complete routing tree connecting the source to all the sinks on this net is built.

C. Effectiveness of the Rerouting Strategy

Experiments are conducted to evaluate the proposed rerouting strategy. Similar to Fig. 1(a), Fig. 1(b) shows the experimental results using VPR improved with the proposed rerouting strategy. As we can see, the proposed rerouting strategy significantly improves runtime by 54.9% over baseline VPR. For more than half of the circuits, the effects on wirelength are negative. On average, critical path delay is reduced by 0.8% and wirelength is slightly increased by 0.6%.

Let T_{vpr} denote the runtime of baseline VPR. We experimentally find that the congested paths accounts for 9.6% of all the paths, and the time spent on routing nets accounts for approximate 86% of the total runtime. Accordingly, if we only reroute the congested paths, the expected runtime is

$$T = 86\% \times T_{\text{vpr}} \times 9.6\% + 14\% \times T_{\text{vpr}} \tag{6}$$

and

$$\left(1 - \frac{T}{T_{\text{vpr}}}\right) \times 100\% = 77.7\%. \tag{7}$$

That is, theoretically, skipping the rerouting of uncongested paths should achieve a 77.7% runtime reduction rather than 54.9% over baseline VPR. The main reason causing the smaller actual runtime reduction is that some additional work has to be done before routing a net. For instance, the improved net router first checks this net's routing tree to rip-up the congested paths, and then performs timing analysis of the current routing tree. In contrast, baseline VPR neither checks the routing tree nor performs timing analysis.

V. Rerouting the Delay-Harmful Paths

In this section, we describe the improvement to net router for the purpose of optimizing the circuit delay. Further, we show that our approach can provide considerable reduction in runtime without sacrificing the quality of results.

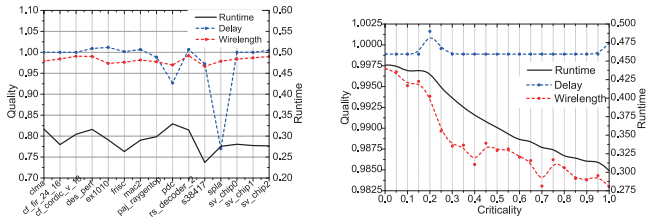


Fig. 2. Comparisons of runtime, critical path delay, and wirelength of baseline VPR to the combination of the (a) improved maze router and the new rerouting strategy, and the (b) proposed approach with different σ values.

When combining the improved maze router and the new rerouting strategy described in Section IV, we evaluate the effects on runtime and the quality of results by the experiments. Fig. 2(a) shows the ratios of runtime, critical path delay, and wirelength of the combining approach to that conducted by baseline VPR. As we can see, the proposed approach keeps the critical path delays of five circuits unchanged, but worsens the delays of more than half of the remaining ten circuits. Unfortunately, skipping the rerouting of uncongested paths has indeed the potential to harm the critical path delay. If the high-criticality path is circuitous but is not congested at present, it will remain unchanged in the current iteration.

In order to optimize the circuit delay, we apply a criticality threshold σ ranged over $\{x|0 \leq x \leq 1, x \in R^{+}\}$ and set up the following two timing-based conditions called timing conditions to pick out the paths that are likely to harm the critical path delay.

1) The criticality of the path is greater than σ.
2) The path's delay is greater than the minimum delay of the paths leading to the same sink built in the prior iterations.

On one hand, the paths satisfying the first condition are likely to be part of the critical path in the current iteration. On the other hand, if a path satisfies the second condition, there exist shorter routing solutions for the target sink. When rerouting a net, not only congested paths, but also the paths satisfying the timing conditions are ripped-up and rerouted. In this way, the paths that may worsen the critical path delay are also rerouted in the current iteration and thus have chances to find better routing solutions.

A. Setting the Criticality Threshold

Obviously, when the criticality threshold σ is smaller, more paths are rerouted. On the contrary, if σ is too large, more paths may lose chances of optimization. In order to achieve a balance between runtime and the quality of results, we set σ to an appropriate value by the experiments. Fig. 2(b) shows the ratios of runtime, critical path delay, and wirelength of our approach to that conducted by VPR with different σ values. As expected, we observe that the runtime of our approach gradually decreases as σ increases. When σ equals 1, the runtime of our approach is about 0.3× the VPR routing runtime. The smaller σ is, the more noncritical paths are rerouted. Unfortunately, a small σ may drive the critical paths to use circuitous routes and thus increase the critical path delay, as shown in Fig. 2(b), the highest delay value appears when σ is around 0.2. Actually, the critical path delay is basically unaffected when σ varies from 0 to 0.9 while it gradually increases when σ is greater than 0.9. Considering the effects of σ on both runtime and the quality of results, we accordingly set σ to 0.9 where the proposed approach has no negative impact on both critical path delay and wirelength.

TABLE I
RUNTIME, CRITICAL PATH DELAY, AND WIRELENGTH FOR TWO CASES: BASELINE VPR 7.0 AND THE PROPOSED RUNTIME-OPTIMIZATION APPROACH

	VPR			HTech			RORA		
Benchmark	**runtime(s)**	**delay(ns)**	**wirelength**	**runtime(s)**	**delay(ns)**	**wirelength**	**runtime(s)**	**delay(ns)**	**wirelength**
cf_cordic_v_18_18_18	3.58	7.55725	28732	1.26	7.55725	28366	1.04	7.55725	28412
cf_fir_24_16_16	10.37	26.9855	79624	3.06	26.9855	78835	3.54	26.9855	78606
clma	11.58	12.5009	74967	4.45	12.5009	73768	3.75	12.5009	73716
des_perf	9.78	7.53632	81471	3.43	7.49468	81005	3.23	7.49468	80729
ex1010	5.12	9.36415	34745	1.91	9.36415	34257	1.6	9.36415	34175
frisc	3.99	12.4181	28652	1.38	12.4181	28391	1.22	12.7278	28052
mac2	13.64	33.2701	88381	4.71	33.2701	88463	4.24	33.2701	86657
paj_raygentop	8.54	9.45348	49507	3	9.45348	48867	2.65	9.34328	48060
pdc	9.11	9.03365	51951	4.1	9.49522	52852	3.32	8.37245	51061
rs_decoder_2	3.37	16.1661	23467	1.35	16.0558	23798	1.3	16.4756	23579
s38417	4.89	7.95561	32681	1.51	7.40471	32062	1.44	7.40471	32043
spla	5.48	10.1564	30707	2.01	8.63449	31182	1.59	7.82125	30429
sv_chip0	16.61	8.01852	103569	6.36	8.01852	103277	5.19	8.01852	102128
sv_chip1	28.73	12.9006	195434	9.32	12.9006	194548	8.23	12.9006	192653
sv_chip2	77.88	22.9832	483312	20.96	23.0934	479427	21.58	23.0934	478297
Geomean	9.24	12.1988	61777	3.24	12.0438	61497	2.91	11.8953	60889
vs. VPR	1	1	1	0.35	0.987	0.996	0.315	0.975	0.986
vs. HTech				1	1	1	0.898	0.988	0.99

VI. EXPERIMENTS

A. Experimental Setup

Experiments ran on an HP workstation running Linux (Fedora-17-i386) with a Intel Xeon X5550 processor and 12 GB of memory. The circuits are selected from the VTR benchmarks [5] and the largest 20 MCNC benchmarks. Circuits are mapped into four-input look up tables (LUTs) using ABC, then packed into configurable logic blocks (CLBs) with ten 4-LUTs and 22 inputs using the VPR packer. The FPGA architecture we used contains unidirectional wire segments spanning two CLBs. Each circuit is routed for ten times and we take the average value of the results. Across all runs, each circuit is routed using a fixed channel width of 1.2× the minimum channel width needed to route the circuit with a maximum of 50 PathFinder iterations. All of the routing tools described in this paper and benchmarks are available for download from http://ictt.xidian.edu.cn/FPGA/.

B. Experimental Results

The following three schemes are compared: 1) "VPR" represents baseline VPR 7.0; 2) "HTech" represents the combination of the two heuristic techniques presented in [1]; and 3) "RORA" represents our approach. The first column of Table I lists the names of circuits, followed by three groups of three columns showing runtime, critical path delay, and total wirelength.

As shown in Table I, our approach significantly reduces routing runtime by 68.5% over baseline VPR while improving the critical path delay and wirelength by 2.5% and 1.4%, respectively. Note that in Fig. 2(a), critical path delays of about half of the circuits are increased compared with VPR 7.0. Comparatively, with the improvement of rerouting the delay-harmful paths, our approach does not have negative impacts on the critical path delays of most of the circuits.

For comparison, Gort and Anderson [6] achieved 40% reduction in router runtime with an extra 6% area overhead. In [1], they report over 3× speed-up using two heuristic techniques but with a increase of 2% in both critical path delay and total wirelength. However, with our approach the experimental results show that routing runtime is reduced by 10.2% on top of the two heuristic techniques presented in [1], while the quality of results is even improved.

VII. RELATED WORK

There is considerable prior work on accelerating the PathFinder routing, such as using variants of sequential algorithms [7], reducing the amount of circuitry [8], and developing parallel versions of routing algorithms [2], [9].

Vahid *et al.* [10] designed a simplified FPGA structure and thus develop simple routing algorithms. The router is 10× faster than baseline VPR. The tradeoff is a 30% degradation in circuit delay and 10% more resource usage. However, many FPGA companies such as Xilinx and Intel are not adapting with the structure considered in [10].

Gort and Anderson [1] put forward two heuristic techniques to accelerate FPGA routing. First, they force a multifanout net to use the same output pin. Second, they only reroutes the congested nets in each iteration. When combined, the two heuristic techniques provide over 3× speedup versus baseline VPR at the cost of 2% increase in both circuit delay and wirelength. The second one bears some similarity to our approach with a key difference: the authors skip the rerouting of uncongested nets, taking the net as the basic unit whereas our approach skips the rerouting of uncongested paths. In this way, our approach avoids the situation in which only a few congested paths cause the rerouting of all sinks on this net.

Lavin *et al.* [8] and Coole and Stitt [11] as well as Frangieh [12], reduced the size of routing problem by using functional reuse, and therefore accelerate FPGA placement and routing. However, the acceleration comes at the expense of performance overhead and limited scalability on account of reduced flexibility in placement and routing. Recently, Shen and Luo [13] parallelized the FPGA routing by using GPU techniques. They reported that their work significantly accelerates the FPGA routing with 2.7% degradation in quality of results. Hoo and Kumar [14] proposed a distributed memory parallel FPGA router called ParaDiMe based on speculative parallelism and path encoding. ParaDiMe speculatively routes net in parallel with careful design of the messaging protocol to reduce the synchronization overhead. They claimed that, compared to VTR, ParaDiMe achieves an average speedup of 19.8× with 32 processes while producing similar quality of results.

VIII. CONCLUSION

In this paper, we present a new approach for accelerating the FPGA routing based on the PathFinder algorithm. With our approach, each

iteration applies a new rerouting strategy rather than reroutes all the sinks. At the lowest level of operation, the maze router initially expands the wavefront out from part of routing tree to search for the target sink. Experimental results show that on average the proposed approach provides a 68.5% runtime reduction compared to baseline VPR without negative impact on the quality of results.

In the future, we will further improve our approach to accelerate the FPGA routing by optimizing the FPGA architecture. Besides, we plan to formalize a parallel routing approach which involves how to resolve the problems of load balancing and synchronization. In addition to reducing runtime, future work will also try to improve the quality of results.

REFERENCES

[1] M. Gort and J. H. Anderson, "Accelerating FPGA routing through parallelization and engineering enhancements special section on PAR-CAD 2010," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 31, no. 1, pp. 61–74, Jan. 2012.

[2] Y. O. M. Moctar and P. Brisk, "Parallel FPGA routing based on the operator formulation," in *Proc. 51st Annu. Design Autom. Conf.*, San Francisco, CA, USA, 2014, pp. 1–6.

[3] M. Gort and J. H. Anderson, "Deterministic multi-core parallel routing for FPGAS," in *Proc. IEEE Int. Conf. Field Program. Technol. (FPT)*, Beijing, China, 2010, pp. 78–86.

[4] C. Mulpuri and S. Hauck, "Runtime and quality tradeoffs in FPGA placement and routing," in *Proc. ACM/SIGDA 9th Int. Symp. Field Program. Gate Arrays*, Monterey, CA, USA, 2001, pp. 29–36.

[5] J. Luu *et al.*, "VTR 7.0: Next generation architecture and CAD system for FPGAS," *ACM Trans. Reconfigurable Technol. Syst. (TRETS)*, vol. 7, no. 2, p. 6, 2014.

[6] M. Gort and J. H. Anderson, "Combined architecture/algorithm approach to fast FPGA routing," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 21, no. 6, pp. 1067–1079, Jun. 2013.

[7] S. Mukherjee and S. Roy, "Graph colouring based multi pin net detailed routing for FPGA using SAT," in *Proc. IEEE 3rd Int. Adv. Comput. Conf. (IACC)*, Ghaziabad, India, 2013, pp. 308–312.

[8] C. Lavin *et al.*, "HMFlow: Accelerating FPGA compilation with hard macros for rapid prototyping," in *Proc. IEEE 19th Annu. Int. Symp. Field Program. Custom Comput. Mach. (FCCM)*, Salt Lake City, UT, USA, 2011, pp. 117–124.

[9] M. Shen and G. Luo, "Accelerate FPGA routing with parallel recursive partitioning," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Austin, TX, USA, 2015, pp. 118–125.

[10] F. Vahid, G. Stitt, and R. Lysecky, "Warp processing: Dynamic translation of binaries to FPGA circuits," *Computer*, vol. 41, no. 7, pp. 40–46, 2008.

[11] J. Coole and G. Stitt, "BPR: Fast FPGA placement and routing using macroblocks," in *Proc. 8th IEEE/ACM/IFIP Int. Conf. Hardw./Softw. Codesign Syst. Synthesis*, Tampere, Finland, 2012, pp. 275–284.

[12] T. Frangieh, "A design assembly technique for FPGA back-end acceleration," Ph.D. dissertation, Dept. Elect. Comput. Eng., Virginia Polytechn. Inst. State Univ., Blacksburg, VA, USA, 2012.

[13] M. Shen and G. Luo, "Corolla: GPU-accelerated FPGA routing based on subgraph dynamic expansion," in *Proc. ACM/SIGDA Int. Symp. Field Program. Gate Arrays*, Monterey, CA, USA, 2017, pp. 105–114.

[14] C. H. Hoo and A. Kumar, "ParaDiMe: A distributed memory FPGA router based on speculative parallelism and path encoding," in *Proc. IEEE Int. Symp. Field Program. Custom Comput. Mach.*, Napa, CA, USA, 2017, pp. 172–179.