Deterministic Parallel Routing for FPGAs based on Galois Parallel Execution Model

Yehdhih Moctar
IBM Research
Austin, USA

Mirjana Stojilović
École Polytechnique Fédérale de Lausanne (EPFL)
School of Computer and Communication Sciences
Lausanne, Switzerland

Philip Brisk
University of California, Riverside
Riverside, USA

***Abstract*—This paper describes a deterministic and parallel implementation of the VPR routability-driven router for FPGAs. We considered two parallelization strategies: (1) routing multiple nets in parallel; and (2) routing one net at a time, while parallelizing the Maze Expansion step. Using eight threads running on eight cores, the two methods achieved speedups of 1.84× and 3.67×, respectively, compared to VPR's single-threaded routability-driven router. Removing the determinism requirement increased these respective speedups to 2.67× and 5.46×, while sacrificing the guarantee of reproducible results.**

I. Introduction

This paper presents a deterministic and parallel implementation of the VPR routability-driven router [1] using Galois [2], [3]. The Galois programming model, compiler, and runtime synergistically accelerate irregular algorithms that dynamically modify linked data structures [2]. Galois implements speculative parallelism, in which multiple threads may concurrently modify the same nodes in a larger data structure (e.g., a graph), and encapsulates conflict detection and resolution in a manner that is fully transparent to the programmer. As such, Galois significantly simplifies the implementation task for irregular algorithms, such as FPGA routers.

Prior work used Galois' non-deterministic execution model to parallelize the Maze Expansion step of the routability-driven router, which routes a single net [4]. This work offers two key extensions. First, we consider the alternative, which is to route multple nets in parallel; second, we implement both approaches using both deterministic and non-deterministic Galois. Our results show that Galois is more successful at exposing parallelism when routing a single net, compared to routing multiple nets concurrently; we also quantify the performance impact of determinism in both cases. With eight threads, our most effective non-deterministic router is $5.46\times$ faster than the single-threaded routability driven router, while our most effective deterministic router is $3.67\times$ faster.

II. FPGA Routing Problem Formulation

VPR is an FPGA architectural modeling and CAD tool framework which is widely used in academic research [5]. VPR features two routers: one which is routability-driven [1], and the other, which is timing-driven [6]; this paper focuses exclusively on the routability-driven router.

FPGA routing is equivalent to the NP-complete problem of finding a set of disjoint paths in a graph (Fig. 1). The primary data structure representing FPGA routing resources is a directed *Routing Resource Graph (RRG)* $G = (V, E)$.

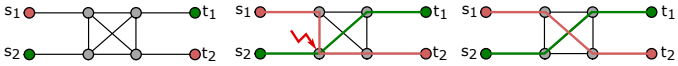

Fig. 1. An instance of the disjoint path problem: (a) a graph with sources $S = \{s_1, s_2\}$ and sinks $T = \{t_1, t_2\}$; (b) an illegal solution, i.e., two non-disjoint paths that share a common vertex; and (c) a legal solution, i.e., two disjoint paths that share no common vertices.

Each vertex $v \in V$ represents a wire or pin and each edge $e \in E$ represents a connection between two vertices. Each signal i to route through G forms a net $N_i = (s_i, \{t_{i,1}, t_{i,2}, \cdots, t_{i,m}\})$. N_i emanates from one source $s_i \in V$ and connects to a set of sinks $\{t_{i,1}, t_{i,2}, \cdots, t_{i,m}\} \subset V$ in G. Let $P_{i,j}$ denote the path from s_i to sink $t_{i,j}$ in G. Two paths $P_{i,j}$ and $P_{i,k}$ that emanate from the same source may overlap, but paths in distinct nets must have disjoint routes, as shown in Fig. 1(c). The solution to the routing problem of net N_i is the directed routing tree $RT(N_i)$.

III. PathFinder Algorithm

VPR's routers are based on a prior algorithm called PathFinder [7]. PathFinder's main body is a triple-nested loop; the outer loop is called the *All-net Router*, the middle loop the *Signal Router*, and the inner loop the *Maze Expansion*:

- The **All-net Router** repeatedly calls the Signal Router to route all of the nets. It terminates either when a legal solution is found, or a user-specified number of iterations fail to produce a legal solution.
- Each **Signal Router** iteration rips up each net and re-routes it by invoking Maze Expansion. Parallelizing the signal router entails routing multiple nets in parallel.
- The **Maze Expansion** traverses the RRG starting from the source of a net N_i. The net's routing tree $RT(N_i)$ is initialized with the source node. Maze Expansion uncovers the neighbors of the source and stores them in a priority queue (PQ), sorted by their cost. Then, it extracts the minimum cost vertex v_{min} from PQ. If v_{min} is a sink, a backtrace procedure constructs a path from the sink to the routing tree and adds the newly created path to the $RT(N_i)$. Otherwise, each undiscovered neighbor v of v_{min} is inserted into the PQ and the Maze Expansion continues. Fig. 2 illustrates an example.

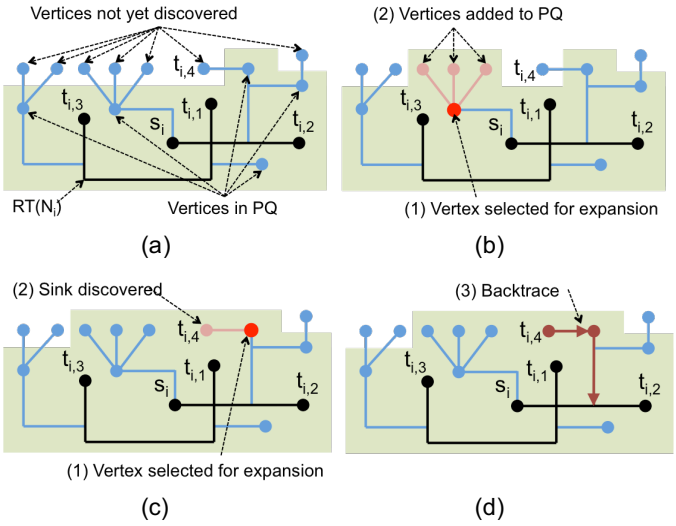

Fig. 2. Maze Expansion: (a) Priority queue PQ contains vertices that have been discovered, but whose neighborhoods have not yet been expanded; $RT(N_i)$ contains part of N_i's routing tree that has been found so far. (b) A vertex is chosen for neighborhood expansion; its neighbors that have not yet been discovered are inserted into PQ, expanding the *wavefront* of the search. (c) Expanding the neighborhood of the next vertex discovers a sink. (d) A backtrace connects the sink to $RT(N_i)$, all vertices and edges along the backtrace are then added to $RT(N_i)$.

During Maze Expansion, nets may share routing resources, creating a temporarily illegal solution. To legalize the result, the All-net Router rips up and re-routes all nets, not just those that share routing resources, and imposes a *penalty cost* on the shared resources to dissuade their use in subsequent iterations; this tends to push the algorithm toward convergence, although convergence is not guaranteed. Typical penalty cost functions consist of at least three terms: $b(v)$, the *base cost* (the intrinsic delay of the routing resource); $p(v)$, the *present congestion* (a first-order congestion term); and $h(v)$ is the *historical congestion* (a second-order congestion term). Refs [7], [1], [6] discuss several penalty function alternatives in great detail.

IV. Irregular Parallelism in PathFinder

The PathFinder Signal Router and Maze Expansion are irregular algorithms with parallelism that can only be discovered at runtime. The Signal Router can route multiple nets in parallel. The common case is that a subset of nets being routed concurrently will not access the same routing resources. However, if two or more nets route through the same vertex v, then a *conflict* occurs. In a serial implementation, routing the first net through v would increase the present congestion cost $p(v)$, which would be read by the Maze Expansion of the second net; this may alter the path that Maze Expansion obtains for the second net, as a lower-cost vertex may be chosen instead. A deterministic parallel router must detect this conflict and delay routing of the second net to ensure that Maze Expansion reads the correct present congestion cost.

Maze Expansion can explore many RRG vertices independently; however, if several concurrent threads discover the same vertex and insert it into their respective priority queues at the same time, then a conflict occurs. A serial implementation would discover each vertex at most once. A deterministic parallel router must ensure that each vertex is discovered via the same parent regardless of the number of threads.

V. Galois

An algorithm in Galois [2] is the repeated application of an *operator* to an element (vertex or edge) of a graph. An element on which a computation is centered is *active*; an *activity* is the application of an operator to an active element; and a *neighborhood* is the set of elements that an activity accesses. Galois may process active vertices in parallel and activities may dynamically spawn other activities.

Each element has an exclusive *lock* that must be acquired by a thread before it can access that element. Locks are held until the activity terminates. If a lock cannot be acquired because it is already owned by another thread, the Galois runtime detects the conflict and *rolls back* one of the conflicting activities.

Iteration coalescing allows a thread to execute multiple loop iterations at once by providing each thread with a *local workset*. An activity that generates new active elements places them in the local workset, instead of the global workset, which is accessed by all threads. When an activity completes, the iteration fetches work from its local workset, if possible, without releasing any locks on the neighborhood. This continues until the local workset is empty, a conflict occurs, or the maximum number of coalesced iterations is reached. The iteration releases its locks when it finishes. If a conflict is detected, the currently executing activity is rolled back, while previous completed coalesced activities commit; the local workset contents are moved into the global workset.

Deterministic Galois [3] constructs an *interference graph*, where vertices correspond to tasks and edges are placed between conflicting tasks. Tasks are executed in rounds, where each round corresponds to an independent set. The scheduler assigns a unique id to each vertex, which enables deterministic heuristics for the independent set problem.

VI. Parallel Routability-Driven Router

A. Parallel Signal Router

Algorithm 1 shows pseudocode for the parallel Galois Signal Router. Let K be the number of threads. The netlist is partitioned into K sets, one set per thread. During the first All-net router iteration, the historical congestion of each RRG vertex is set to zero. Subsequent iterations rip-up and re-route the result of the previous iteration and increase historical congestion costs as needed. All threads share a record of successfully routed nets that the Galois runtime has committed. Each worker thread routes its nets serially and updates intermediate routing results through lock-based data structures in shared memory. Worker threads synchronize their respective views of the routing state and synchronize upon completion of the iteration. Speculation conflicts occur when distinct worker threads concurrently route multiple nets through the same RRG vertex. The Galois runtime rolls back the misspeculated

Algorithm 1 Pseudocode of VPR's routability-driven router where the **signal router** has been parallelized using Galois. **Inputs:** Set of nets to route $N = \{N_i | 1 \leq i \leq n\}; RRG : G = (V, E)$. **Outputs:** Set of routing trees $R = \{RT(N_i) | 1 \leq i \leq n\}$. **Variables:** integer $iter\_cnt$, boolean $found\_sink$, boolean $v\_discovered$.

```
1:  iter_cnt ← 1
2:  for all vertices v ∈ V do
3:      v_discovered ← false
4:      initialize_vertex_cost(v)
5:  end for

6:  // Global router
7:  while iter_cnt < MAX_ITER do
8:      Partition N into K sets
9:      n_i = the number of nets allocated to the i^th thread
10:     N_{i,j} = the j^th net allocated to the i^th thread
11:     for all i = 1 to K in parallel do
12:         for all j = 1 to n_i do
13:             for all vertices v ∈ RT(N_{i,j}) \ N_{i,j} do
14:                 p(v) ← update_present_congestion(v)
15:             end for
16:             Rip-up routing tree RT(N_{i,j})
17:         end for
18:     end for

19:     // Parallel signal router
20:     for all i = 1 to K in parallel do
21:         for all j = 1 to n_i do
22:             sequential_maze_expansion(G, N_{i,j})
23:             update_congestion_costs(G, N_{i,j})
24:             backtrace(G, N_{i,j})
25:         end for
26:     end for

27:     // Threads synchronize here;
28:     // post-processing for the global iteration
29:     for all nets N_i do
30:         for all vertices v ∈ RT(N_i) do
31:             h(v) ← update_historical_congestion(v)
32:         end for
33:     end for
34:     iter_cnt ← iter_cnt + 1
35: end while
```

Algorithm 2 Pseudocode of VPR's routability-driven router where the **maze expansion** has been parallelized using Galois. **Inputs:** Set of nets to route $N = \{N_i | 1 \leq i \leq n\}; RRG : G = (V, E)$. **Outputs:** Set of routing trees $R = \{RT(N_i) | 1 \leq i \leq n\}$. **Variables:** Integer $iter\_cnt$, boolean $found\_sink$, boolean $v\_discovered$, global priority queue GPQ, array of vertex costs $VCost$.

```
1:  Allocate RRG, GPQ, R and VCost in shared memory

2:  iter_cnt ← 1
3:  for all vertices v ∈ V do
4:      v_discovered ← false
5:      initialize_vertex_cost(v, VCost[v])
6:  end for

7:  // Global Router
8:  while (iter_cnt < MAX_ITER and there exists at least
        one congested vertex in the RRG) do

9:      // Signal Router
10:     for all i = 1 to n do
11:         for all vertices v ∈ RT(N_i) \ N_i do
12:             update_present_congestion(v, VCost[v])
13:         end for
14:         Rip-up routing tree RT(N_i)
15:         RT(N_i).insert(s_i)
16:         for all threads T do
17:             // Assume iteration coalescing
18:             Allocate a local priority queue T.LPQ
19:             Connect T.LPQ to GPQ
20:             // Galois maze expansion relies on the Galois
21:             // runtime system to acquire locks, speculatively
22:             // execute operators and commit/abort operators
23:             // as appropriate
24:             T.Galois_maze_expansion(G, N_i, VCost)
25:         end for
26:     end for

27:     for all nets N_i do
28:         for all vertices v ∈ RT(N_i) do
29:             update_historical_congestion(v, VCost[v])
30:         end for
31:     end for
32:     iter_cnt ← iter_cnt + 1
33: end while
```

routing trees and waits until the conflicting worker thread completes its route; then the misspeculated worker threads proceed. This ensures that the present congestion of the RRG reflects the route computed by the conflicting worker thread before the misspeculated worker threads reroute their nets.

B. Parallel Maze Expansion

Algorithm 2 presents pseudocode for the parallel Galois Maze Expansion. With iteration coalescing enabled, each thread has a local priority queue (LPQ) and shared memory holds a global priority queue (GPQ). Each thread accesses its LPQ to read the next vertex, only accessing the GPQ when its LPQ is empty. The *active elements* are the vertices in the LPQs; the *neighborhoods* are the sets of vertices adjacent to each active vertex; the *operator* is the neighborhood expansion which inserts newly discovered adjacent vertices into the LPQs. When a sink is found, the backtrace procedure involves a different set of active elements, neighborhood definition, and operator. Maze Expansion stops when all sinks are found.

TABLE I
FPGA ARCHITECTURAL PARAMETERS.

K	N	W	I	F_{cin}	F_{cout}	CLB Area
6	10	1.4 W_{min}	33	0.15	0.1	8069.46

TABLE II
SUMMARY OF THE IWLS 2005 BENCHMARKS USED HERE.

Benchmark circuit	FPGA size	Nets	CLBs
ac_ctrl	48 × 48	5097	5008
aes_core	33 × 33	5800	2518
des_area	16 × 16	1569	695
mem_ctrl	27 × 27	4464	3158
pci_bridge32	74 × 74	8016	7815
spi	13 × 13	923	712
systemcaes	21 × 21	2509	2173
systemcdes	12 × 12	1068	706
usb_funct	40 × 40	5154	4429
wb_conmax	47 × 47	10430	6297

The GPQ, RRG, and routing trees for each net are stored in shared memory along with an array, $VCost$, which contains the relevant cost terms associated with each RRG vertex. $VCost$ entries are stored separately from the RRG to reduce contention for locks. Results are reported using a non-blocking PQ based on *software transactional memory (STM)* [8].

VII. EXPERIMENTAL SETUP

We ported VPR into the Galois system, making sure that all data structures were thread-safe. We used VPR 5.0 [5] to compare with prior work [4] and used the same benchmarks (10 of the largest circuitrs from IWLS 2005 [9]). Tables I and II respectively list the FPGA architectural parameters and benchmarks. We used ABC [10] for logic synthesis and technology mapping, T-VPack for placement, and compared directly with VPR's routability-driven router [1].

For each benchmark, we compute W_{min} using VPR's routability-driven router (sans parallelization). Routing experiments for each benchmark are then performed using channel width $W = 1.4 \times W_{min}$. The maximum number of PathFinder iterations MAX_ITER set to 100; we found legal routing solutions in all of our experiments. We placed each benchmark using ten different random number seeds; we routed each placed circuit three times. For each benchmark and parallelization strategy, we report the average execution time of all thirty runs. All experiments were performed on a server featuring 8 Intel Xeon E5540 processors running at 2.53 GHz and 40 GB shared memory. We ran our router using 1, 2, 4, and 8 threads, for both the non-deterministic and deterministic schedulers [3].

VIII. EXPERIMENTAL RESULTS

Fig. 3(a)-(d) reports the speedups of the parallelized Signal and Maze Routers. The baseline is VPR's sequential routability-driven router with no modifications.

With eight threads running on eight cores, deterministic Galois achieved average speedups of 1.84× for the Signal Router and 3.67× for Maze Expansion; non-deterministic Galois achieved respective speedups of 2.67× and 5.46×. Most notably, the slowdown incurred by deterministic Galois Maze Expansion compared to its non-deterministic counterpart increased with the number of threads (18% for two threads; 33% for eight threads). Thus, deterministic execution may not scale well under the Galois model.

The Galois Signal Router acquires locks for each routing tree $RT(N_i)$ before committing. This creates large undo lists, which increases the cost of conflict resolution when partial routing trees must be discarded; this also negatively impacts load balancing among threads. The cost to roll back a conflicting operation (i.e., discovering a vertex) during Maze Expansion is less than tearing down a partially routed net. Iteration coalescing reduces the number of accesses to lock-based shared data structures and improves load balancing: if a worker thread can obtain work from the GPQ whenever its LPQ is empty. The Galois Maze Expansion handles these performance bottlenecks better than the Galois Signal Router, which explains the performance differences reported in Fig. 3.

IX. RELATED WORK

Most parallel FPGA routers parallelize PathFinder, often with modifications to ensure determinism or to optimize performance [11], [4], [12], [13], [14], [15]. Alternatives include linear programming with Lagrangian relaxation [16] and using Bellman-Ford in lieu of Maze Expansion, which is amenable to parallelization using a GPU [17]; although both of these papers achieve substantial speedups compared to a single-threaded CPU, they also report significantly degraded solution quality.

PathFinder's result depends on the order in which nets are routed [18]. Several parallel routers alter the net ordering in order to achieve higher performance; however, doing so leads to non-deterministic results [4], [14], [15]. In contrast, deterministic routers incur overhead due to thread synchronization or limit parallel routing to nets with non-overlapping bounding boxes [11], [12], [13], [17].

X. CONCLUSION AND FUTURE WORK

The Galois programming model is ideal for irregular algorithms, such as FPGA routers, because it encapsulates the underlying details of speculative parallelism, such as lock acquisition, speculation conflicts, and rollback, from the programmer. A more recent update to the Galois runtime offers deterministic execution, which is of great importance to industry. This paper parallelized the VPR routability-driven router using Galois and two different parallelization strategies, and quantified the performance disparity between deterministic and non-deterministic execution. Beyond parallelization, this paper made no changes to the VPR routability-driven router. In principle, algorithmic enhancements proposed by others that are compatible with the Galois Signal Router and/or Galois Maze Expansion could be added to either implementation; that said, evaluating the performance of these enhancements in the context of deterministic and/or non-deterministic Galois is not the primary objective of this paper. It is certainly possible that additional enhancements to the Galois Signal Router could make it more competitive with Galois Maze Expansion.

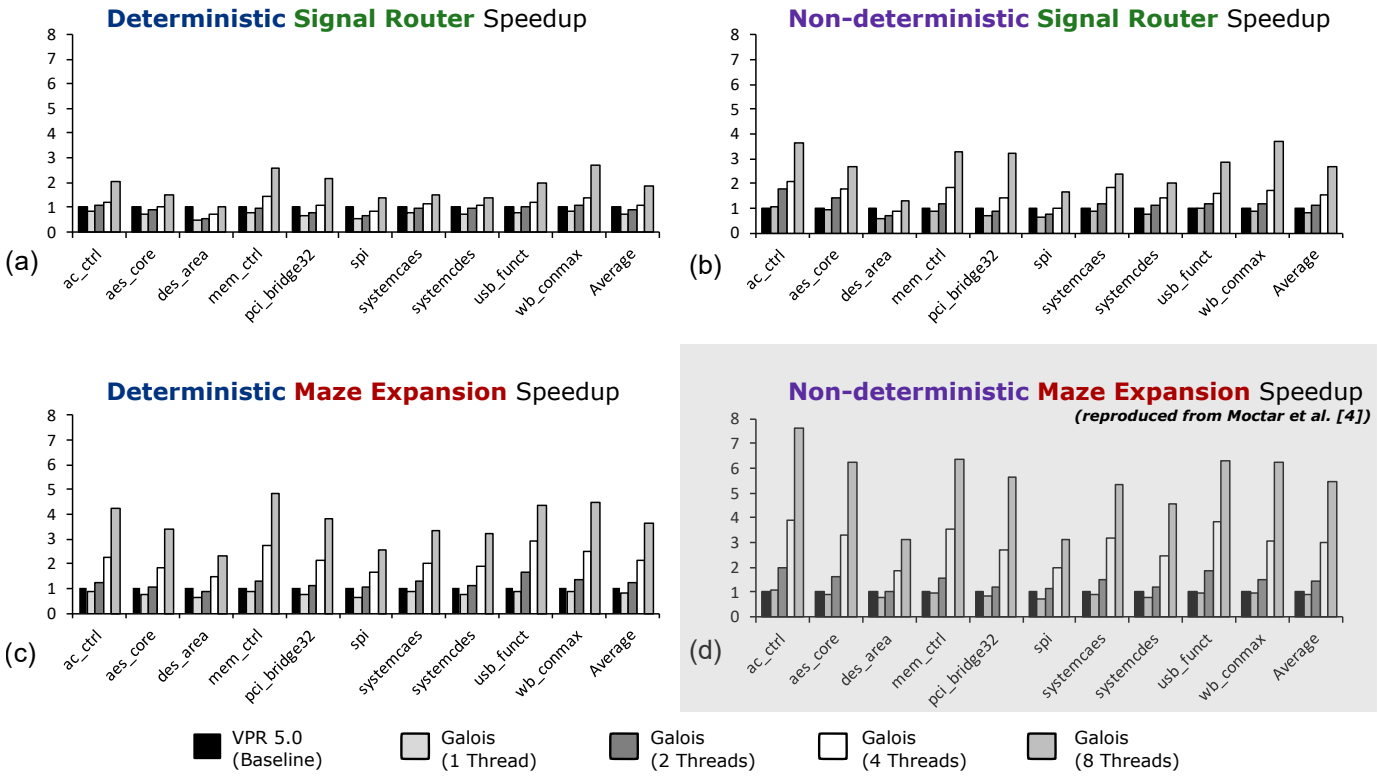

Fig. 3. Speedup results normalized to the single-threaded VPR 5.0 routability-router using Galois with 1, 2, 4, and 8 threads. (a) Galois Signal Router, deterministic scheduler; (b) Galois Signal Router, non-deterministic scheduler; (c) Galois Maze Expansion, deterministic scheduler; and (d) Galois Maze Expansion, non-deterministic scheduler (reproduced from Ref. [4]).

Acknowledgment

This work was supported in part by NSF Award #1528181.

References

[1] J. S. Swartz, V. Betz, and J. Rose, "A fast routability-driven router for FPGAs," in *Proc. of the 6th ACM/SIGDA Intel. Symp. on Field Programmable Gate Arrays*, Monterey, Calif., Feb. 1998, pp. 140–49.

[2] K. Pingali, D. Nguyen, M. Kulkarni, M. Burtscher, M. A. Hassaan, R. Kaleem, T.-H. Lee, A. Lenharth, R. Manevich, M. Méndez-Lojo, D. Prountzos, and X. Sui, "The tao of parallelism in algorithms," in *Proc. of the ACM SIGPLAN'11 Conf. on Programming Language Design and Implementation*, San Jose, Jun. 2011, pp. 12–25.

[3] D. Nguyen, A. Lenharth, and K. Pingali, "Deterministic Galois: On-demand, portable and parameterless," in *Proc. of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, Salt Lake City, Utah, Mar. 2014, pp. 499–512.

[4] Y. O. M. Moctar and P. Brisk, "Parallel FPGA routing based on the operator formulation," in *Proc. of the 51st Design Automation Conference*, San Francisco, Calif., Jun. 2014, pp. 1–6.

[5] J. Luu, J. H. Anderson, and J. Rose, "Architecture description and packing for logic blocks with hierarchy, modes and complex interconnect," in *Proc. of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, Monterey, Calif., Feb. 2011, pp. 227–36.

[6] V. Betz, J. Rose, and A. Marquardt, *Architecture and CAD for Deep-submicron FPGAs*. Boston, Mass.: Kluwer Academic, 1999.

[7] L. McMurchie and C. Ebeling, "PathFinder: A negotiation-based performance-driven router for FPGAs," in *Proc. of the 3th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, Monterey, Calif., Feb. 1995, pp. 111–17.

[8] K. Dragicevic and D. Bauer, "A survey of concurrent priority queue algorithms," in *Proc. of the 22nd IEEE International Parallel and Distributed Processing Symposium*, Miami, FL, Apr. 2008, pp. 1–6.

[9] IWLS, "Benchmark suite," 2005. [Online]. Available: http://iwls.org/iwls2005/benchmarks.html

[10] Berkeley logic synthesis and verification group, "ABC: A system for sequential synthesis and verification," 2005. [Online]. Available: http://www.eecs.berkeley.edu/~alanmi/abc

[11] M. Gort and J. H. Anderson, "Accelerating FPGA routing through parallelization and engineering enhancements," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 1, pp. 61–74, Jan. 2012.

[12] C. Zhu, J. Wang, and J. Lai, "A novel net-partition-based multithread FPGA routing method," in *Proc. of the 23rd International Conference on Field-Programmable Logic and Applications*, Porto, Portugal, Sep. 2013, pp. 1–4.

[13] M. Shen and G. Luo, "Accelerate FPGA routing with parallel recursive partitioning," in *Proc. of the International Conference on Computer Aided Design*, Austin, Texas, Nov. 2015, pp. 118–25.

[14] C. Hau Hoo, Y. Ha, and A. Kumar, "ParaFRo: A hybrid parallel FPGA router using fine grained synchronisation and partitioning," in *Proc. of the 26th International Conference on Field-Programmable Logic and Applications*, Lausanne, Aug. 2016, pp. 1–11.

[15] C. Hau Hoo and A. Kumar, "ParaDiMe: A distributed memory FPGA router based on speculative parallelism and path encoding," in *Proc. of the 25th IEEE Symposium on Field-Programmable Custom Computing Machines*, Napa Valley, Calif., Apr. 2017, pp. 172–79.

[16] C. Hau Hoo, A. Kumar, and Y. Ha, "ParaLaR: A parallel FPGA router based on Lagrangian relaxation," in *Proc. of the 25th International Conference on Field-Programmable Logic and Applications*, London, Sep. 2015, pp. 1–6.

[17] M. Shen and G. Luo, "Corolla: GPU-accelerated FPGA routing based on subgraph dynamic expansion," in *Proc. of the 25th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, Monterey, Calif., Feb. 2017.

[18] R. Y. Rubin and A. M. DeHon, "Timing-driven pathfinder pathology and remediation: Quantifying and reducing delay noise in VPR-pathfinder," in *Proc. of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, Monterey, Calif., Feb. 2011, pp. 173–76.