ROUTING ALGORITHMS FOR FPGAS WITH SPARSE INTRA-CLUSTER ROUTING CROSSBARS

Yehdhih Ould Mohammed Moctar[1] *Guy G. F. Lemieux*[2] *Philip Brisk*[1]

[1]Dept. of Computer Science and Engineering
University of California, Riverside
{moctar, philip}@cs.ucr.edu

[2]Dept. of Electrical and Computer Engineering
University of British Columbia
lemieux@ece.ubc.ca

ABSTRACT

Modern FPGAs employ sparse crossbars in their intra-cluster routing. Modeling these crossbars enlarges the routing resource graph (RRG), a data structure used by most FPGA routers, while enlarging the search space for finding legal routes. We introduce two scalable routing heuristics for FPGAs with sparse intra-cluster routing crossbars: SElective RRG Expansion (SERRGE), which compresses the RRG, and dynamically decompresses it during routing, and Partial Pre-Routing (PPR), which locally routes all nets in each cluster, and routes global nets afterwards. Our experiments show that: (1) PPR and SERRGE converge faster than a traditional router using a fully-expanded RRG; (2) they both achieve better routability than the traditional router, given a limited runtime budget, with SERRGE achieving 1-2% better routability than PPR, on average; and (3) PPR uses far less memory and runs much faster than SERRGE, making it ideal for high capacity FPGAs.

1. INTRODUCTION

FPGA routing is the process by which paths through the FPGA routing fabric are selected to establish connections between logic elements and the I/O pads. To establish a connection, the router must find a sequence of unused resources along a path from the source to the sink.

The Routing Resource Graph (RRG) is a data structure that represents the routing resources of an FPGA. Routing a circuit is equivalent to finding a set of disjoint paths in the RRG that satisfy timing constraints; this problem is NP-hard [19]. The most successful FPGA routing heuristic is the PathFinder algorithm [19], which iteratively tries to eliminate congestion while minimizing the delay of the critical paths.

The academic Versatile Place and Route (VPR) tool [17] employs an implementation of PathFinder. VPR models a cluster-based architecture, in which LUTs are grouped into Configurable Logic Blocks (CLBs); connections between LUTs in a CLB use fast intra-cluster routing, which is implemented as a crossbar.

Most early cluster-based FPGAs used a full crossbar for intra-cluster routing; however, the area overhead of a full crossbar is significant. Modern cluster-based FPGAs now employ sparse crossbars for intra-cluster routing. In the past, a router could simply route all nets to CLB input pins, as a full crossbar ensures the existence of paths from each CLB input pin and local feedback to each LUT input pin; however sparse crossbars do not provide this guarantee, so the router must now consider the sparse crossbar topology at each CLB. Naïvely, one could expand the RRG to include the sparse crossbar for each CLB, as routing would remain a disjoint path problem; however, doing so enlarges the search space for finding legal and high quality routes, and the expanded RRG consumes more memory.

This paper introduces and compares two new routing algorithms for FPGAs with sparse intra-cluster routing crossbars. SElective RRG Expansion (SERRGE) executes PathFinder on a compressed RRG; SERRGE decompresses regions of the RRG where PathFinder is actively exploring and negotiating new routes. Partial Pre-Routing (PPR) first runs PathFinder at every CLB to route the intra-cluster portion of all nets, and then runs PathFinder globally to complete the inter-cluster portion of the routes. PPR's advantages are its simplicity and the fact that it never expands the RRG; its limitation is that it cannot re-negotiate intra-CLB routes that collectively cause congestion in the global routing network.

We observed that running PathFinder on a fully expanded RRG is not competitive with either SERRGE or PPT in terms of runtime or memory usage; PPR is faster and uses less memory than SERRGE and produces comparable results in terms of critical path delay, except for very sparse crossbars, e.g., 40% population density. Based on these experiments, we believe that PPR should be used in most cases, and that SERRGE should be employed as a backup, primarily when PPR either fails to route the circuit or fails to meet timing constraints.

2. BACKGROUND AND TERMINOLOGY

This section briefly reviews the FPGA architecture employed by VPR [17], as shown in Fig. 1.

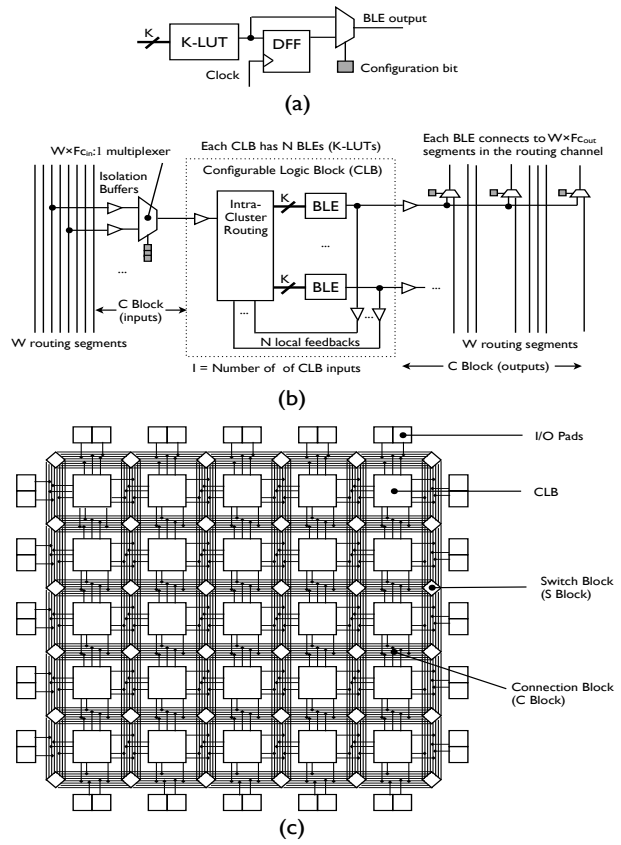

Fig. 1. Basic Logic Element (BLE) (a); a Configurable Logic Block (CLB) contains several BLEs, with fast local interconnect provided by the Input Interconnect Block (IIB); the Connection Block (C Block) inputs and outputs interface the CLB with the global routing network (b); the floorplan of a generic island-style FPGA (c).

The basic unit of computation in an FPGA is a K-input LookUp Table (K-LUT). A Basic Logic Element (BLE) is a K-LUT coupled with a bypassable flip-flop, as shown in Fig. 1(a).

Fig. 1(b) shows the CLB interface to the adjacent routing channel. The Connection Block (C Block) connects each CLB input pin to a subset of the wires in the adjacent routing channel; the intra-cluster routing connects the CLB input pins and local feedbacks (one per BLE) to the BLE inputs.

Fig. 1(c) shows the FPGA floorplan. Switch Blocks (S Blocks) are programmable intersections between horizontal and vertical routing channels. The multiplexers, shown on the right-hand side of Fig. 1(b), are implemented in the S Blocks, which are shown (without detail) in Fig. 1(c). Fig. 1(b) depicts inputs coming in from the left hand side of the CLB and outputs leaving to the right; in actuality, inputs and outputs may enter and exit from all four sides.

VPR's architecture configuration file specifies several parameters that VPR uses to generate the logic and routing architecture of an FPGA under exploration:

K: the LUT size (i.e., a K-LUT);
N: the number of LUTs per CLB;
I: the number of CLB input pins;
W: the number of segments per routing channel; and
Fc_{in} and Fc_{out}: C Block connectivity parameters

Each C Block input multiplexer in Fig. 1(b) selects one of $W \times Fc_{in}$ wires, and each BLE drives $W \times Fc_{out}$ segments in the adjacent routing channels. Most FPGAs use single driver routing [13], so the C Block output is a conceptual description of the routing topology.

We model the intra-cluster routing as a 2-dimensional binary matrix *B*, with *I+N* columns and *KN* rows. Each column corresponds to an input (a CLB input pin or a local feedback from a BLE in the cluster), and each row corresponds to a BLE input. *B(i, j) = 1* if a signal can route from input *i* to output *j*, and *0* otherwise. It is important to note that *B* simply models the input-to-output connectivity of the crossbar, but does not model its internal architecture.

As an example, we model a CLB with *N=2*, *K=2* (e.g., it contains two 2-LUTs); the four BLE inputs are denoted b_{00}, b_{01}, b_{10}, and b_{11}. The CLB has three input pins, I_0, I_1, and I_2, and two local feedbacks from the BLEs, O_0 and O_1:

$$B = \begin{array}{c} \begin{matrix} I_0 & I_1 & I_2 & O_0 & O_1 \end{matrix} \\ \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{bmatrix} \end{array} \begin{matrix} b_{00} \\ b_{01} \\ b_{10} \\ b_{11} \end{matrix}$$

In this example, there is a connection from CLB input pin I_0 to LUT input pins b_{00} and b_{01}, but not b_{10} and b_{11}; also, the local feedbacks are not used.

We used a tool developed by Lemieux et al. [15] to generate routable sparse crossbars with a user-provided density function *p*. The tool generates matrix *B* such that each row, column, and the entire matrix all have a population percentage of approximately *p*, i.e.:

$$\sum_j B(i,j) = \lceil p(I+N) \rceil \pm 1, \quad (1)$$

$$\sum_i B(i,j) = \lceil pkN \rceil \pm 1, \text{ and} \quad (2)$$

$$\sum_{i,j} B(i,j) = \lceil pk(I+N) \rceil \pm 1. \quad (3)$$

3. FPGA ROUTING ALGORITHMS

The first step of most routers is to allocate the RRG. In VPR's implementation of PathFinder, RRG vertices represent wires and pins internal to the FPGA, and edges represent the switches that connect them. Here, we describe the assumptions that underlie VPR's router, along with two heuristic variations that reduce the RRG's memory footprint. These variations restrict the search space explored by the router, while speeding up convergence.

3.1. CLB Input Pins

VPR 5 (and its predecessors) assumes that the intra-cluster routing is a full crossbar; all CLB input pins are equivalent and there is no need to model the intra-cluster routing. A legal route is obtained by routing all circuit nets to CLB input pins, as the full crossbar guarantees trivially-found routes from CLB input pins to LUT input pins.

When the intra-cluster routing becomes sparse, CLB input pins are no longer equivalent; some may be equivalent, depending on the crossbar topology. Recall that I is the number of CLB input pins, and that N is the number of BLEs in the CLB. We group the CLB input pins into N non-disjoint subsets of size r_j, $0 \leq j \leq N\text{-}1$ under the assumption that the input pins of each subset are logically equivalent. Subset S_j contains the CLB input pins that can be routed to BLE input pin j. Based on this assumption, BLE j can select its inputs from S_j and S_j only, and there are $r_j = |S_j|$ inputs available for BLE j. It is important to note that these sets of input pins are non-disjoint, i.e., most CLB input pins will belong to multiple subsets.

The tool that we used to generate the sparse crossbar topology for the intra-cluster routing tries to balance the fanout of each CLB input pins to approximately $p(kN)$ [15]. Thus, all CLB input pins are approximately equivalent in terms of critical path delay and their general connectivity.

3.2. Baseline Routing Algorithm

The Baseline router expands the RRG with extra vertices and edges to represent the topology of the intra-cluster routing for each CLB. Pathfinder now routes nets to BLE input pins, rather than CLB input pins, and must therefore be augmented with information about which BLE is the sink of each net. Beyond that, all input pins of the same BLE are treated as equivalent.

One drawback is that the RRG becomes significantly larger: the search space for legal and high quality routes grows, and the larger data structure may stress the memory subsystem of the computer that runs the router. VPR's implementation of PathFinder is otherwise unmodified.

3.3. Routing with Selective RRG Expansion (SERRGE)

In the baseline router, the RRG requires the usage of memory to store both the FPGA architectural parameters and data about the route that PathFinder is presently computing. The physical FPGA parameters are stored statically, while the temporary routing data is dynamic.

For the static routing resources, any optimization must alter the way that VPR represents the physical layout of the FPGA. This would require a major retooling of VPR, and has been left open for future exploration. Instead, we try to reduce the overhead of dynamic memory resources required by VPR's implementation of PathFinder.

To reduce the memory footprint of the RRG, we modified VPR's implementation of PathFinder to perform Selective RRG Expansion (SERRGE). SERRGE features a custom memory manager and garbage collector that are specific to the RRG and other associated data structures used by VPR's implementation of PathFinder.

VPR's PathFinder implementation routes one sink at a time; nets with fanout have multiple sinks. PathFinder's wave expansion step finds a path from the source to an input pin j of the CLB that contains the sink. Upon finding a CLB input pin, SERRGE then expands the RRG to include the neighborhood of the CLB input pin, i.e. the LUT input pins driven by the CLB input pin. This expansion facilitates negotiation with the LUT input pins as future nets are routed. As long as the selected CLB input pin yields a quality route, SERRGE does not expand the RRG to include the entire sparse crossbar topology.

We also introduced a garbage collector that periodically inspects the size of the RRG; if the dynamically-allocated RRG grows more than 30% larger than its original size (i.e., the RRG that represents the global routing resources, but no intra-cluster routing), then the garbage collector is invoked to delete all non-essential data from the dynamically expanded RRG, expansion maps, and trace-back arrays. In practical terms, this ensures that the dynamically generated RRG is never more than 1.3x larger than the RRG used for FPGAs with fully-populate intra-cluster routing crossbars (i.e., the RRG used in VPR 5.0, which does not support sparse crossbars).

3.4. Routing with Partial Pre-Routing (PPR)

Routing with Partial Pre-Routing (PPR) is a two-step process that is conceptually simpler than SERRGE, although theoretically more constrained. The PPR stage performs intra-CLB routing first, followed by a global routing step, which is constrained by the PPR result.

The PPR step is applied to each CLB in the FPGA that contains at least one sink. PPR routes nets from CLB input pins to BLE inputs, from BLE output pins to CLB output pins, and from BLE output pins to BLE input pins through the intra-cluster routing's local feedbacks.

PPR models this problem as a multi-commodity flow, where the set of sources are the CLB input pins, the sinks are the BLE input pins, and the intra-cluster routing wires are intermediate vertices in the RRG. The demand represents the number of CLB input pins from which each BLE can select its inputs, and is essentially a function of the intra-cluster routing crossbar. We invoke PathFinder to route the nets locally within the CLB.

In the degenerate case where no nets in a CLB have fanout, then the problem becomes a single commodity flow that can be solved in polynomial time, e.g., using a network flow algorithm; we have not implemented this optimization thus far, as PPR already runs efficiently (Section 5.2).

After PPR, PathFinder is invoked to perform global routing, i.e., from input pads to CLBs, between CLBs, and from CLBs to output pads. This step is conceptually similar to FPGA routing with full crossbar intra-cluster routing; however, the PPR solution at each CLB imposes additional constraints. With full crossbar intra-cluster routing, all CLB input pins are equivalent, as each CLB input pin connects to each BLE input pin within the CLB. PPR, in contrast, connects specific CLB input pins to specific BLE input pins. Therefore, all $k \leq K$ CLB input pins that connect to the same BLE input are equivalent; that is, to say, the k nets that drive the k inputs of the LUT within the BLE can be routed to the k CLB input pins in any order; however, *none* of the other CLB input pins connect to that specific BLE, as a result of running PPR as a preprocessing step.

Routing with PPR partitions the RRG into smaller disjoint graphs, which are routed individually. If we assume that all CLBs have the same intra-cluster routing crossbar topology, then PPR can generate one RRG up-front and re-use it for local routing within all CLBs, followed by a second RRG for the inter-CLB router. Since the lifetimes of both RRGs are disjoint, the memory requirement of PPR should not exceed that of a router for an FPGA with fully-populated intra-cluster routing crossbars.

4. EXPERIMENTAL SETUP

The purpose of our experiments is to evaluate the performance and memory consumption of routing with SERRGE (Section 3.3) and PPR (Section 3.4) in comparison to the baseline router (Section 3.5).

4.1. VPR 5.0

We began this work when VPR 5.0 [17] was the most up-to-date version and we have implemented our routing approaches in that framework. VPR 5.0 did not support sparse intra-cluster routing; we added the required features, as discussed in Section 2. We used a tool described by Lemieux and Lewis [15] to generate routable sparse crossbars with a user-specified population density.

We used ABC [2] for logic synthesis and technology mapping, T-VPack[1] for packing, and VPR 5.0 for placement and routing. We modified the architectural configuration system for VPR 5.0 to introduce sparse intra-cluster routing in the form of the B matrix (see Section 2). We modified VPR's implementation of PathFinder to accommodate sparse intra-cluster routing crossbars as described in Sections 3.2-3.4. We attempted to retain VPR 5.0's timing-driven features as much as possible throughout our implementation.

[1] T-VPack has since been deprecated as part of the Verilog-to-Routing (VTR) flow; packing is now integrated into VPR 6.0.

The Beta release of VPR 6.0 [16] featured sparse intra-cluster routing, but did not include a timing-driven router; that feature was added to the official release of VPR 6.0 early in 2012, when the implementation work outlined here was mostly complete. VPR 6.0's router appears to be similar to routing with PPR (Section 3.4), however, we have not compared against it directly.

4.2. Experimental Parameters

For the FPGA architectural configuration, we took four of the VPR architecture files from the iFAR repository [10, 11]. We considered architectures with LUT sizes ranging from 4-7. Table 1 lists the baseline parameters for the architectures we considered.

We considered intra-cluster routing crossbar population densities of 40%, 50%, 75%, and 100% in our experiments. We vary several of the architectural parameters, including N (the number of LUTs per cluster) and K (LUT size). As suggested in ref. [1], we set the number of inputs pins per CLB to $I = ½K(N+1)$. The I/O pad capacity was set to 8.

VPR repeatedly routes each benchmark using a binary search to identify the smallest channel width, W_{min}, for which a legal route can be found. VPR also allows the user to specify a chosen channel width (W), and VPR will try its best to find a legal route, but may fail. We took that latter approach in our experiments, and set $W = 1.4W_{min}$.

PathFinder takes an iterative approach to routing, and terminates after a user-specified number of iterations. We set the maximum number of iterations allowed to 100; if PathFinder cannot find a successful route after 100 iterations, then we assume that it has failed.

4.3. Timing and Area Models

Our timing model was similar to VPR 5.0. We added models to account for delays inside of the CLBs. The timing graph is generated such that every CLB or LUT input pin becomes a timing node. The connectivity between pins is represented by timing edges, and delays are marked on edges, rather than nodes.

The area model sums the aggregate areas of the number of minimum-width transistors required to place and route a circuit on an FPGA. We did not modify the counting method used in VPR, but we did add extensions to account for the intra-cluster routing area, which now depends on the population density of the internal crossbar. We employed the basic techniques that were used in VPR to estimate the silicon area occupied by each MUX and wire in the CLB. We assume that a minimum width transistor takes 1 unit of area. A double-width transistor takes twice the diffusion width, but the same spacing, so we assume it takes 1.5x the area of a minimum-width transistor.

Table 1. FPGA architectural parameters taken from configuration files in the iFAR repository [10, 11]. 65nm CMOS (BPTM) was assumed for all architectures.

K	N	W	I	F_{cin}	F_{cout}	CLB Area
4	10	96	22	0.2	0.1	3956.25
5	10	96	28	0.15	0.1	5496.54
6	10	96	33	0.15	0.1	8069.46
7	10	104	39	0.15	0.1	13065.9

Buffer sizes are calculated based on the drive strength requirements and depend on the fan-out of the buffer. VPR uses 4x the minimum size, which we have adopted for general buffers. We sized the CLB input buffers using the approach used by Lemieux et al. [14], where the drive strength is at least 7x and at most 25x the minimum size.

We model an FPGA with single-driver wires; each wire segment begins with a multiplexer followed by a driver. We attempt to judiciously select the multiplexer size depending on the number of inputs. One-level multiplexers are used when there are 4 or fewer inputs, and more levels are used when the number of multiplexer inputs increases.

4.4. Benchmarks

We selected 10 of the largest IWLS benchmarks [17] for use in our experiments; Table 2 summarizes them.

VPR generates a custom FPGA that is sized for each benchmark. The second column of Table 4 lists the dimensions of the FPGA generated for each benchmark (e.g., an MxN array of CLBs). The third and fourth columns list the number of nets and CLBs used in each benchmark for an FPGA architecture with parameters *N=8*, *K=6*, and *I=27*, i.e., each CLB contains eight 6-LUTs and has 27 input pins.

5. EXPERIMENTAL RESULTS

All experiments here compare SERRGE (Section 3.3) and PPR (Section 3.4) against the baseline router (Section 3.2), and against one another. Since we model FPGAs with sparse intra-cluster routing crossbars, we cannot compare directly against VPR 5.0's implementation of PathFinder, except for the special case of a fully-populated crossbar.

Table 2. Benchmark summary.

Benchmark	Array size	Nets	CLBs
ac_ctrl	48x48	5097	5008
aes_core	33x33	5800	2518
des_area	16x16	1569	695
mem_ctrl	27x27	4464	3158
pci_bridge32	74x74	8016	7815
spi	13x13	923	712
systemcaes	21x21	2509	2173
systemcdes	12x12	1068	706
usb_funct	40x40	5154	4429
wb_conmax	47x47	10430	6297

All experiments are run three times per benchmark, using different random number seeds for placement; each placed circuit is then routed, and measurements are taken from the results of the routing: e.g., routability, runtime, critical path delay, etc. The results are then averaged across the ten IWLS benchmarks, unless stated otherwise.

We limited PathFinder to 100 routing iterations. Our routability experiments (Section 5.1) report the percentage of nets that were routed legally. Our other experiments—runtime (Section 5.2), memory consumption (Section 5.3), and critical path delay (Section 5.4)—only include results for benchmarks that routed successfully. For example, we *do not* include the critical path delay of a benchmark that failed to route in our averaged results.

5.1. Routability

Figs. 2 and 3 report the percentage of nets routed without any sharing violations after 100 iterations of PathFinder, targeting an FPGA with sparse intra-cluster routing crossbars with a population density of *p=50%*.

In Fig. 2, the CLB size is fixed at *N=8* BLEs, and the LUT sizes vary from *K=4* to *7*; in Fig. 3, the LUT size is fixed at *K=6*, and the number of BLEs per CLB varies from *N=4* to *10*.

SERRGE and PPR complete a greater percentage of their routes (after 100 iterations of PathFinder) than the Baseline router; the results for the baseline router degrade significantly for 6- and 7-LUTs, and for CLBs with 8 or 10 BLEs. In most cases, SERRGE routes 1-2% more nets successfully than PPR. With more routing iterations, Baseline could potentially obtain improved routability, as it would have more time to explore a larger search. SERRGE and PPR converge faster because they restrict the search space that is explored.

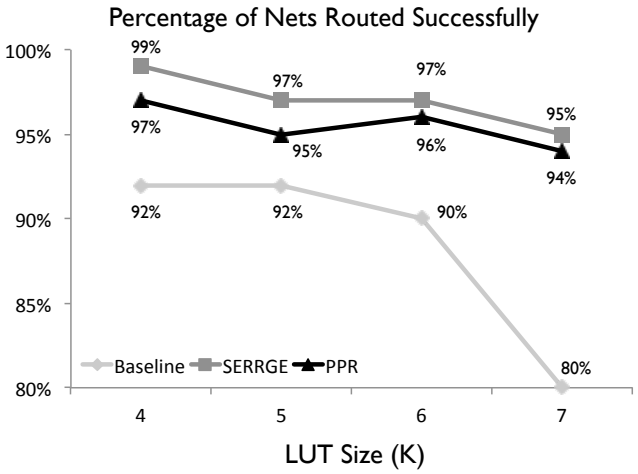

Figure 2. Percentage of nets routed successfully on FPGAs after 100 routing iterations, with LUT sizes ranging from 4 to 7. Here N = 8, p = 50%, and I = 18, 23, 27, and 32 respectively.

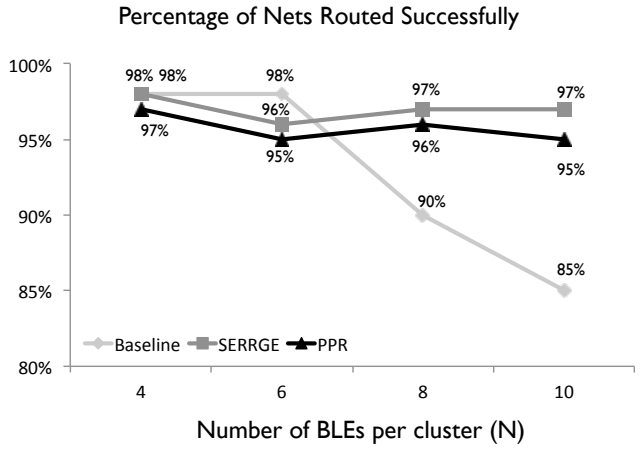

Figure 3. Percentage of nets routed successfully on FPGAs after 100 routing iterations, with various CLB sizes N = 4, 6, 8, and 10. Here K=6, p = 50%, and I = 15, 21, 27, and 33 respectively.

5.2. Runtime

Our second experiment compares the runtimes of the routing algorithms as a function of intra-cluster routing crossbar population density (*p*). As shown in Fig. 4, the runtime of all three routing algorithms increases monotonically with *p*: the baseline has the largest runtime, as expected; SERRGE is in the middle; and PPR runs the fastest, and shows the greatest robustness to variations in *p*.

For *p = 100%*, it makes sense to compare against the VPR 5.0 PathFinder implementation, which only works for fully populated intra-cluster routing crossbars, rather than the Baseline router (Section 3.2). Fig. 5 reports the result of this comparison, varying the LUT size (*K*). PPR and VPR 5.0's router achieve comparable runtimes, while SERRGE runs considerably slower.

Fig. 6 reports the runtime of the intra-cluster routing phase of PPR as a function of *p*; the reported runtimes increase monotonically with *p*; however, even in the worst case (*p = 100%),* the runtimes of the intra-cluster routing phase of PPR are on the order of tens of seconds, while the overall runtime of PPR plus global routing is on the order of hundreds of seconds (Fig. 5). Thus, the runtime of PPR is quite robust to variations in *p*.

5.3. Memory Consumption

Fig. 7 reports the static and dynamic memory consumption of the Baseline router, SERRGE, and PPR for FPGAs with intra-cluster routing crossbars with population densities of *p = 40%* and *100%*. The dynamic memory consumption reported in Fig. 7 is the maximum amount of allocated data (in megabytes) taken over the runtime of the router. These results are averaged over all of the benchmarks.

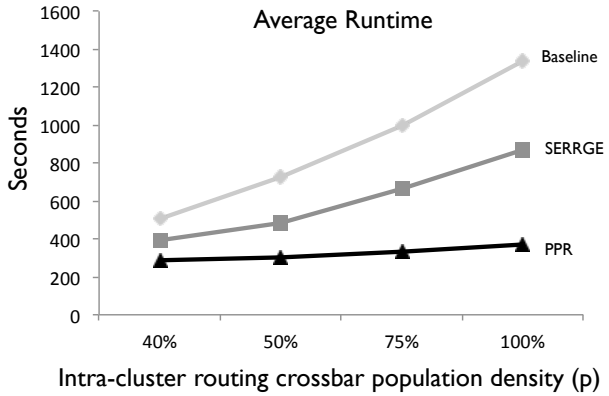

Figure 4. Average runtime of the routers with varying population densities. In this experiment, N=8, K=6, and I=27.

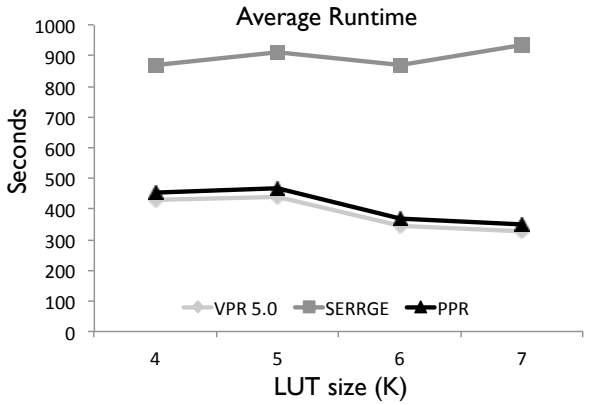

Figure 5. Average runtime of PPR, SERRGE, and the VPR 5.0 router for FPGAs with intra-cluster routing crossbar population densities of p=100%. Here N=8, I=27, and K=4, 5, 6, and 7.

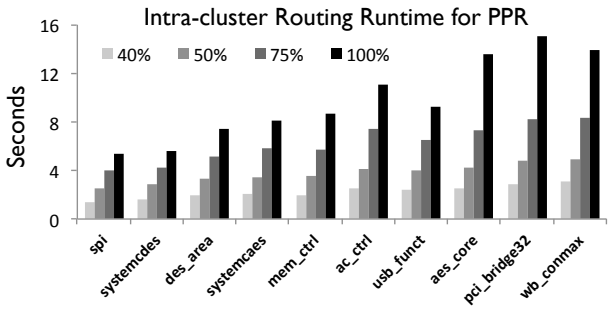

Figure 6. Intra-cluster routing runtime for PPR for different population densities. Here K=6, N=8, and I=27.

SERRGE and PPR use considerably less static and dynamic memory than the Baseline router, while PPR uses marginally less memory than SERRGE. It is important to note that the memory overhead of SERRGE is managed explicitly by the threshold used by the garbage collector, and can be tuned up or down.

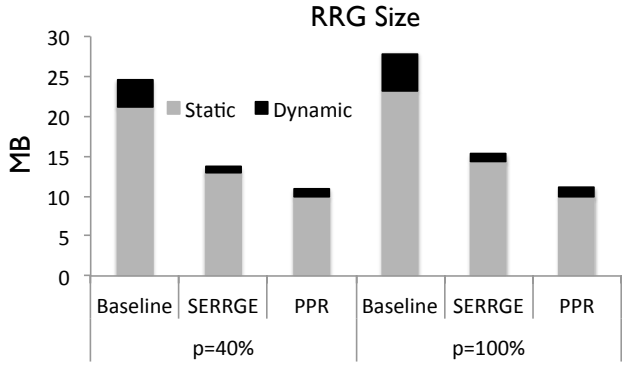

Figure 7. Average static and dynamic memory consumption of the three routers for FPGAs with sparse intra-cluster routing crossbars with population densities p=40% and p=100%. Here K=6, N=8, and I=27.

5.4. Critical Path Delay

Aside from routability, critical path delay is an important metric by which to evaluate a router. Fig. 8 reports the average critical path delay obtained by the Baseline router, SERRGE, and PPR as a function of intra-cluster routing crossbar population density, *p*.

In Fig. 8, the largest reported variation between routers is *0.55ns* (*p = 75%*), which is not particularly significant. The general trend suggests that the Baseline Router achieves critical path delays that are around 0.5ns slower than SERRGE or PPR; however, there is also a lot of variance in quality between SERRGE and PPR: SERRGE is better in two cases (*p = 40%* and *100%*), worse in one case (*p = 75%*), and both PPR and SERRGE are equal in the last case (*p = 50%*). We conservatively conclude that the three algorithms are comparable to one another in terms of critical path delay. This could be further explored using experimental noise-reduction strategies [20]; we leave this particular issue open for future work.

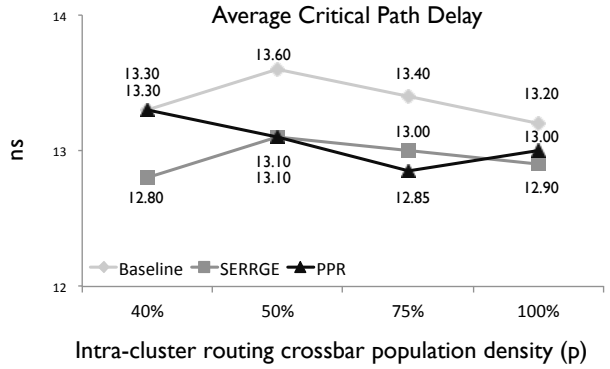

Figure 8. Average critical path delay of the three routers for FPGAs with sparse intra-cluster routing crossbars with population densities p=40%, 50%, 75%, and 100%. Here K=6, N=8, and I=27.

6. RELATED WORK

Lemieux et al. [15] introduced the idea of sparse crossbars in 2000, and fully evaluated their use as intra-cluster routing one year later [14]. Later work by Feng and Kaptanoglu [7] use entropy counting to design input crossbars that lead to interconnect topologies with greater routability.

Lemieux and Lewis [14] suggested that routability could be improved by adding spare CLB input pins. The goal of this work is comparable, however, we improve routability through the introduction of effective CAD algorithms, without requiring any architectural changes.

Chin and Wilton [6] developed techniques to reduce the memory requirements of routing algorithms, which influenced the development of SERRGE significantly. Their work did not attempt to model sparse crossbar topologies within the intra-cluster routing of a CLB; thus, their work does not address the problem of RRG expansion, which occurred in our Baseline router (Section 3.2). Similarly, other routers that have been published over the past 20 years (e.g., refs. [4, 5, 12]) are certainly relevant to this work, but address different issues.

The PathFinder routing algorithm [19] has been at the core of this work, due to its implementation in VPR and its enduring legacy. It is important to recall that neither SERRGE nor PPR has attempted to replace PathFinder, and, they both use it as a subroutine. Their objectives were simply to avoid expanding the RRG, and, in doing so, they restrict the search space provided to PathFinder when it is called, in order to achieve faster convergence and reduce memory overhead.

7. CONCLUSION

The SERRGE and PPR routing algorithms were introduced to cope with the runtime and memory overhead associated with introducing sparse intra-cluster routing crossbar information into an FPGA's RRG. The Baseline router, which extends the RRG with intra-cluster routing topology information for each CLB, runs slowly and consumes an inordinate amount of memory.

SERRGE is considerably more complex to implement than PPR, as it is essentially an application-specific dynamic memory management and garbage collection framework that has been specialized to meet the needs of the RRG and the PathFinder routing algorithm. Although we did not discuss details, significant modifications were made to several of VPR 5.0's internal data structures in order to facilitate these new features.

In contrast, PPR is much simpler, as it computes routes within each CLB up-front, and then computes a global route that obeys those constraints imposed by the pre-computed intra-CLB routes. Thus, our implementation and debugging effort for PPR was far less than for SERRGE.

PPR routes faster than SERRGE and requires less memory; in its favor, SERRGE achieved routability improvements of 1-2% for different intra-cluster routing crossbar population densities in comparison with PPR. Both SERRGE and PPR restrict the search space explored by PathFinder as it routes a circuit through the RRG. In particular PPR fixes all intra-cluster routes prior to global routing, which can lead to global failures if demand for a specific subset of global wires is particularly high.

We believe that it should be possible to combat this limitation by invoking a SERRGE-inspired recovery step, which would facilitate re-negotiation of intra-CLB routing resources, when PPR fails; a detailed investigation of this enhancement is left open for future work.

8. REFERENCES

[1] E. Ahmed, and J. Rose, "The effect of LUT and cluster size on deep submicron FPGA performance and density," *IEEE Trans. VLSI*, vol. 12, no. 3, pp. 288-298, Mar. 2003. DOI= http://dx.doi.org/10.1109/TVLSI.2004.824300

[2] Berkeley Logic Synthesis and Verification Group. "ABC: A system for sequential synthesis and verification.: Dec. 2005 release." URL= http://www.eecs.berkeley.edu/~alanmi/abc

[3] V. Betz, and J. Rose, "Automatic generation of FPGA routing architectures from high-level descriptions," in *Proc. ACM/SIGDA Int. Symp. FPGAs*, Feb. 2000, pp. 175-184. DOI= http://doi.acm.org/10.1145/329166.329203

[4] S. Brown, J. Rose, and Z. G. Vranesic, "A detailed router for field programmable gate arrays," *IEEE Trans. on Computer-Aided Design*, vol. 11, no. 5, pp. 620-628, May 1992. DOI= http://dx.doi.org/10.1109/43.127623

[5] Y. W. Chang, K. Zhu, and D. F. Wong, "Timing-driven routing for symmetrical array-based FPGAs," *ACM Trans. Design Automation of Electronic Systems*, vol. 5, no. 3, pp. 433-450, Jul. 2000, DOI= http:/doi.acm.org/0.1145/348019.348101

[6] S. Y. L. Chin, and S. J. E. Wilton, "Static and dynamic memory footprint reduction for fpga routing algorithms," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 1, no. 4, pp. 1–20, Jan. 2009. DOI= http://doi.acm.org/10.1145/1462586.1462587

[7] W. Feng, and S. Kaptanoglu, "Designing Efficient Input Interconnect Blocks for LUT Clusters Using Counting and Entropy," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 1, no. 1, pp. 1-28, Mar. 2008. DOI= http://doi.acm.org/10.1145/1331897.1331902

[8] IWLS 2005 Benchmarks. URL= http://iwls.org/iwls2005/benchmarks.html

[9] J. Rose, J. Luu, C. Yu, O. Densmore, J. Geoders, A. Sommerville, K. B. Kent, P. Jamieson, and J. Anderson, "The VTR project: architecture and CAD for FPGAs from Verilog to routing," in *Proc. ACM/SIGDA Int. Symp. FPGAs*, Feb. 2012, pp. 77-86. DOI= http://doi.acm.org/10.1145/2145694.2145708

[10] I. Kuon, and J. Rose, "Area and delay trade-offs in the circuit and architecture design of FPGAs," in *Proc. ACM/SIGDA Int. Symp. FPGAs* Feb., 2008, pp. 149-158. DOI= http://doi.acm.org/10.1145/1344671.1344695

[11] I. Kuon, and J. Rose, "Automated transistor sizing for FPGA architecture exploration," in *Proc. ACM/IEEE Design Automation Conference*, June 2008, pp. 792-795. DOI= http://doi.acm.org/10.1145/1391469.1391671

[12] Y. S. Lee, and C. H. Wu, "A performance and routability-driven router for FPGAs considering path delay." In *Proc. ACM/IEEE Design Automation Conference*, June 1995, pp. 557–561. DOI= http://dx.doi.org/10.1109/DAC.1995.250009

[13] G. Lemieux, E. Lee, M. Tom, and A. Yu, "Direction and single-driver wires in FPGA interconnect," in *Proc. IEEE Int. Conf. Field-Programmable Technology*, Dec. 2004, pp. 41-48. DOI= http://dx.doi.org/10.1109/FPT.2004.1393249

[14] G. Lemieux, and D. Lewis, "Using sparse crossbars within LUT clusters," in *Proc. ACM/SIGDA Int. Symp. FPGAs*, Feb. 2001, pp. 59-68. DOI= http://doi.acm.org/10.1145/360276.360299

[15] G. Lemieux, P. Leventis, and D. Lewis, "Generating highly-routable sparse crossbars for PLDs," In *Proc. ACM/SIGDA Int. Symp. FPGAs*, Feb. 2000, pp. 155-164. DOI= http://doi.acm.org/10.1145/329166.329199

[16] J. Luu, J. Anderson, and J. Rose, "Architecture description and packing for logic blocks with hierarchy, modes and complex interconnect," in *Proc. ACM/SIGDA Int. Symp. FPGAs*, Feb.-Mar. 2011, pp. 227-236. DOI= http://doi.acm.org/10.1145/1950413.1950457

[17] J. Luu, I. Kuon, P. Jamieson, T. Campbell, A. Ye, W. M. Fang, and J. Rose, "VPR 5.0: FPGA CAD and architecture exploration tools with single-driver routing, heterogeneity and process scaling," in *Proc. ACM/SIGDA Int. Symp. FPGAs*, Feb. 2009, pp. 133-142. DOI= http://doi.acm.org/10.1145/1508128.1508150

[18] A. Marquardt, V. Betz, and J. Rose, "Using cluster-based logic blocks and timing-driven packing to improve FPGA speed and density," in *Proc. ACM/SIGDA Int. Symp. FPGAs*, Feb. 1999, pp. 37-46. DOI= http://doi.acm.org/10.1145/296399.296426

[19] L. McMurchie, and C. Ebeling, "Pathfinder: a negotiation-based performance-driven router for FPGAs," in *Proc. ACM/SIGDA Int. Symp. FPGAs*, Feb. 1995, pp. 111-117. DOI= http://doi.acm.org/10.1145/201310.201328

[20] R. Rubin and A. DeHon, "Timing-driven Pathfinder pathology and remediation: quantifying and reducing delay noise in VPR-Pathfinder," in *Proc. ACM/SIGDA Int. Symp. FPGAs*, Feb. 2011, pp. 173-176. DOI= http://dx.doi.org/10.1145/1950413.1950447