Timing-Driven Placement for FPGA Architectures with Dedicated Routing Paths

Stefan Nikolić*, Grace Zgheib†, Paolo Ienne*
*École Polytechnique Fédérale de Lausanne (EPFL)
School of Computer and Communication Sciences, 1015 Lausanne, Switzerland
{stefan.nikolic, paolo.ienne}@epfl.ch
†Intel Corporation
San Jose, USA
grace.zgheib@intel.com

***Abstract*—The idea of introducing dedicated, fast paths between certain FPGA elements in order to reduce delay is neither new nor particularly hard to come up with. What is less obvious, however, is how to put such paths to actual use. In this work, we propose an effective ILP-based detailed placer for FPGA architectures with direct connections between LUTs. We discuss various aspects of making such an approach practicable, from efficient formulation of the integer programs themselves, to focused application of the placer on specific portions of the circuit where it could have the greatest impact. These careful considerations allow us to simultaneously move tens of LUTs with tens of candidate positions each, in a matter of minutes. This more than doubles the advantage of additional connections on the critical path delay compared to the previously reported results that relied on architecture-oblivious placement algorithms.**

I. Introduction

Introducing dedicated connections between some *Field-Programmable Gate Array* (FPGA) resources in order to reduce their delays, by avoiding multiple levels of multiplexing in the programmable routing architecture, is an old idea [3], [9]. A problem in exploring such architectures is that there could be two different causes for failing to achieve the anticipated effect of the additional connections. We could, for instance, expect that a cascade of *Look-Up Tables* (LUTs) is reasonably useful in reducing the critical path delay of a typical circuit. If we fail to observe any benefit, a logical guess could be that our CAD tools do not provide adequate support for such cascades. Before dedicating effort to envisioning new algorithms, it would be useful to know that the problem does not lie in the simplicity of the cascade itself, because, for instance, it cannot cover multiple fanouts or fanins. Unfortunately, without having an optimal algorithm for putting the cascades to use, we cannot be sure that this is really the case. Hence, a lack of good algorithms makes it hard to assess the quality of the architectures, while the lack of a good architecture makes it hard to assess the quality of the algorithms.

It has been recently shown, however, that there exists at least one architecture which profits from the additional connections even if no special CAD tools are used to map circuits onto them [18]. The only CAD flow change used to demonstrate the effectiveness of this architecture consisted of shuffling the LUTs within their respective clusters, to align them with the endpoints of the direct connections, after the circuit has been placed. This changes the perspective considerably, because it is no longer a question of which architecture with fast connections is useful, but if its usefulness can be increased by application of CAD tools that are aware of the existence of the fast connections.

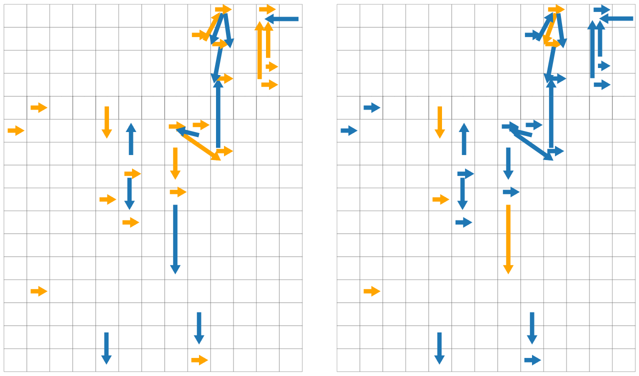

Fig. 1: Influence of movement freedom on delay minimization. Arrows depict a subset of connections of the *or1200* circuit, placed by standard VPR, on the architecture of Figure 2. The blue arrows mark the connections that are successfully implemented as direct after appropriately moving their endpoint LUTs, while those that remain programmable are shown in orange. In the left figure, LUTs can only move within their cluster, while in the right, they can also move to the adjacent clusters, resulting in a 900 ps smaller delay.

Intuitively, one would expect that the latter question is easier to resolve than the former one and this is what inspired the present work. Before diving into details, it is useful to quantify our hopes. Previous work reports a 3% improvement of the average critical path delay of a subset of *VTR* benchmarks [13] [18]. If the delays of all the connections between the LUTs were reduced to the average delay of the direct connections in the proposed architecture, the improvement would rise to about 19%. This is clearly not achievable, but it shows that there is likely a fairly big margin for improvement.

A more illustrative example is given in Figure 1. The arrows show a subset of connections of the *or1200* circuit, placed by standard *VPR* [13] on an architecture with a 60-input cluster of ten 6-LUTs and the pattern of direct connections shown in Figure 2. The resulting initial postplacement delay is 13 ns. Moving the LUTs within their respective clusters, in an attempt to improve this delay by aligning the depicted connections with the direct connections of the architecture, produces the figure on the left. The blue connections are the ones successfully aligned, resulting in a delay of 12.57 ns. Allowing the LUTs to move to the adjacent clusters as well produces the situation

978-1-7281-9902-3/20/$31.00 ©2020 IEEE
DOI 10.1109/FPL50879.2020.00035

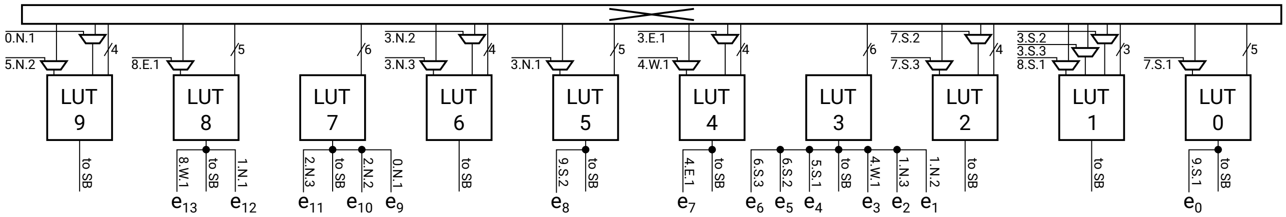

Fig. 2: The target architecture [18]. The other endpoint of each direct connection is labeled as $L.O.D$, where $L \in (0, 9)$ is the index of the LUT in its cluster, $O \in \{N, S, E, W\}$ is the connection orientation with respect to the shown cluster, and D the distance to the other cluster.

on the right, with the delay of 11.67 ns. These numbers were produced by actual optimization, as described in the following sections, and clearly demonstrate the benefits of moving LUTs across clusters.

In this paper, we investigate the impact of dedicated placement on performance of architectures with direct connections between LUTs. To do so, we develop an efficient placement algorithm that is specifically tailored towards such architectures. Our general approach to the placement problem consists of (i) selecting a specific subset of LUTs to be moved and then (ii) solving an *Integer Linear Program* (ILP) to perform their actual placement; we present it in Section II. Sections III and IV describe the details of LUT-subset selection and ILP formulation, respectively. A way to integrate these two parts into a complete placement flow is presented in Section V. Experimental setup and results are reported in Sections VI and VII, respectively, while Section VIII concludes the paper.

II. General Approach

In this section, we describe our basic approach to placement for FPGAs with dedicated fast connections, such as the architecture shown in Figure 2. The architectural assumptions are that the direct connections connect individual LUTs, but that they only provide alternative, faster routing paths, and that the router can always opt for general routing instead. In other words, any placement that is legal for the underlying architecture on which the direct connections are instantiated continues to be legal for the complete architecture with the direct connections. Any architecture satisfying these assumptions is supported by the algorithms presented here.

Prior work demonstrated about 3% critical path delay improvement due to direct connections and a rough upper bound for the same circuits presented in the introduction of this paper is 19%. Any number in this interval is but a small fraction of the improvement achieved by a general placer, compared to an unoptimized, random placement. Hence, rather than performing dedicated placement from the very beginning, we start from the solution provided by a general placer and adopt a detailed placement approach, focusing on direct connections only once their impact becomes significant in the bigger picture.

A. Relation to Prior Work

There is an abundance of published work introducing detailed placers, both for FPGAs and ASICs [1], [5], [6], [10], [11], [16], [17]. Most of them operate on a sliding-window principle, where a fixed region of the chip is selected for optimization and then iteratively changed by sliding the window that determines it [14]. One basic distinction between the various algorithms is how they optimize inside the window. Some of them rely on heuristics [5], [10] while others use exact optimization methods, such as ILP [1], [11], SAT [17], or SMT [16]. The virtue of heuristics lies in their scalability which allows them to target larger windows at once, possibly increasing the improvement margin. Exact methods are usually not as scalable, so they are confined to smaller windows, with possibly smaller improvement margin, but are guaranteed to actually meet it.

We take a slightly different approach. Instead of using sliding windows to guide local optimization, we preselect a number of LUTs, regardless of their starting physical location, in such a way that their movement results in maximum delay improvement. This removes the spatial constraints that may be artificial, while still allowing for tight control of the problem size. A similar approach has been used in the previous architectural study, when performing local positioning of LUTs within their respective clusters, but we completely replace the ad hoc method of selecting the LUTs to be moved [18]. An example of spatial distribution of movable nodes obtained in this way can be seen in Figure 1. Each of the cells in the figure is a cluster of ten LUTs, that is, ten potentially movable objects. A sliding-window approach would likely be limited to not more than a few cells in width and height [14]. There are works that directly try to optimize the critical paths [6], which also alleviates the spatial restrictions. As will become apparent in Section III, our method is more general. Instead of being limited to simple paths, it automatically extracts subcircuits of arbitrary complexity.

Direct connections are very sparse in a typical architecture considered here, thus requiring a high level of precision in placing the LUTs if the right connections of the circuit are to be aligned with them. This leaves heuristics with less space for doing a good-enough job at various points in the circuit that would accumulate to a large net improvement than they could have had if the direct connections were a more abundant resource. For this reason, instead of attempting to design elaborate heuristics, we choose the exact approach. In particular, we opt for using ILP, due to the ease of modeling the timing information. Yet, we formulate the placement problem itself in a way that can be easily converted to SAT. In this work, we treat the solver as a black box and focus on developing an efficient formulation of the ILPs.

The necessity to precisely position individual LUTs increases the potential number of movable nodes as well as candidate locations for each of them by an order of magnitude when compared to the classical problem of placing entire clusters [6].

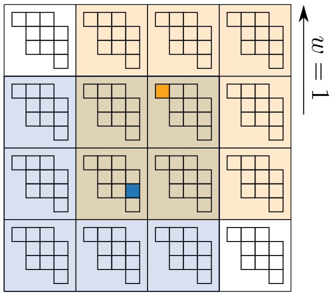

Fig. 3: Scope of LUT movement. A portion of the FPGA grid is shown, with two LUTs highlighted. They can move to any position in any cluster inside the square of half-width w, centered at their initial cluster.

However, as we will see shortly, it is exactly the sparsity of the dedicated interconnect that will help us resolve this problem, by enabling more efficient formulations than in the case of general detailed placement [17].

III. The LP-Based Node Selector

The first step in our placement flow is to determine the LUTs that will be moved from their initial positions. This problem is fundamentally linked to determining which connections of the circuit should have their delays reduced so that the reduction of the critical path delay is maximized.

A. Scope of LUT Movement

Figure 3 shows a portion of the FPGA grid with two LUTs highlighted. Each LUT that is selected for movement can move to any cluster within the square of half-width w centered at its initial cluster. It can be placed at any position in these clusters. All other LUTs remain stationary as do the hard IPs and other similar blocks. During the placement process, movable LUTs are allowed to be placed at positions already occupied by the stationary ones. Such overlaps are only removed in a postprocessing step described in Section V. The rationale is that improving the delay of the connections between the selected movable nodes outweighs any damage caused by moving the remaining nodes so as to achieve a legal packing.

B. Which Connections Should be Improved?

Let $\tau_{u,v}$ be the initial postplacement delay of the connection $e = (u, v)$, as determined by the general placer. Let $t_{u,v}$ be its final postplacement delay and $imp_{u,v}$ the difference between the initial and the final delay. Representing the delay of each connection in the circuit as $t_{u,v} = \tau_{u,v} - imp_{u,v}$ and solving the following *Linear Program* (LP) due to Hambrusch and Tu [8] gives the minimum critical path delay if the total connection delay improvement is not larger than some constant M:

$$\begin{aligned} \min \quad & ta_{max}, && && (1)\\ \text{s.t.} \quad & ta_u \leq ta_{max}, && \forall u \in V, && (2)\\ & ta_v \geq ta_u + t_{u,v}, && \forall (u,v) \in E, && (3)\\ & \sum_{(u,v)\in E} imp_{u,v} \leq M, && && (4)\\ & 0 \leq imp_{u,v} \leq I_{u,v}, && \forall (u,v) \in E. && (5) \end{aligned}$$

The ta variables designate the node arrival times, while constraints (2)–(3) model timing. Constants $I_{u,v}$, bounding the individual connection delay improvements, are obtained from the delays of the fastest direct connections that can implement them, given the movement restrictions imposed on their endpoints. The values assigned to the imp variables in the solution describe one possible distribution of the delay improvement budget M among the connections of the circuit that, if met, guarantees the minimum critical path delay ta_{max}. Throughout this work, we assume that delays can improve only due to connections being implemented as direct and not due to becoming shorter, for example.

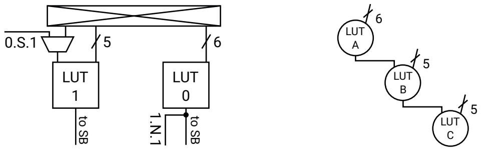

Fig. 4: Pitfalls of the basic selection LP. An architecture is shown on the left, and a piece of a circuit on the right. A solution selecting both circuit connections for improvement is valid, but not supported by the architecture in which each LUT has only one incident direct connection.

C. Determining Movable Nodes

To extract the movable nodes from the solution of the above LP, we simply introduce a threshold θ on the minimum delay improvement and consider only the edges with $imp_{u,v} \geq \theta$ as candidates for implementation with direct connections. The nodes incident to them are the movable ones.

Keeping the number of movable nodes reasonably small is crucial to being able to solve the integer program modeling their movement. We can approximate the number of movable nodes by tuning the delay improvement budget M. However, if we have an upper bound on the critical path delay, we can impose it as a bound on ta_{max} in the above LP and minimize M instead. While neither approach can guarantee the minimum cardinality of the movable node set, due to the fractional nature of the delay improvements, we at least avoid the approximation trouble. This is the approach that we take in this work. Details on obtaining the bound are given in Section V.

D. Specialization of the Selection LP to the Architecture

Solving the selection LP to obtain the set of movable nodes does not yet give a sufficient condition for meeting the desired critical path delay. What is missing is successfully solving the related placement ILP, whose formulation we will discuss in Section IV. This placement ILP may not be feasible, however. For instance, simultaneous improvement of two different connections might imply two nodes being placed at the same position or one node being placed at two distinct positions at once. Aside from the initial placement of the circuit and the allowed movement regions, the FPGA architecture strongly influences feasibility of the placement ILPs constructed from the solutions of the selection LPs. Figure 4 shows a simple architecture and a piece of a circuit. With the current selection LP formulation, there is nothing that would prevent the solution from including both edges of the circuit, although it is clear that the architecture will not be able to improve both of them.

We cannot enforce exclusivity in choosing between these two edges without introducing integer variables. We can, however, use additional constraints to stimulate solutions that the architecture can support. To begin with, we can introduce a bound on the sum of the improvements of the two edges, equal to the maximum of the two individual bounds. This still does not prevent the solution from including both connections, but covering only one of them during the placement process will suffice for this short path to meet what is expected of it in terms of overall delay reduction. For that reason, we introduce pairwise improvement bounds, for each pair of edges sharing a common node. In general, this will not be the maximum of the two individual bounds, but the largest total improvement achievable within the movement regions of the three incident nodes. To further improve feasibility, we include bounds on the total improvement of the incoming, the outgoing, and all the edges incident to each individual node.

IV. The ILP-Based Placer

In this section, we discuss various aspects of formulating the ILP that models the movement of the nodes selected by the process described in the previous section.

A. Naive ILP Formulation

Let V_m denote the set of movable nodes, E_c the set of edges chosen for improvement, and E_s the set of edges in $E \setminus E_c$ that have at least one node in V_m. The edges in E_c are those which caused the nodes to be moved in the first place: they need to be covered by direct connections for the desired delay to be met. The edges in E_s are the ones that potentially suffer degradation due to one of their endpoints being moved.

Each LUT of the FPGA can be described by a triple (x, y, i), where x and y are the coordinates of its cluster and i the index within it. These LUTs are valid positions for the nodes of V_m. Let $P(u, w)$ be the set of positions within the square bounded by w, centered at the initial cluster of u (Figure 3). To each node $u \in V_m$, we can assign the following set of variables: $x_{u,p} \in \{0, 1\}, \forall p \in P(u, w)$. The variable $x_{u,p}$ is 1 iff node u is placed at position p. The following set of constraints describes a valid placement, where we again note that overlaps with nodes outside V_m are removed in a postprocessing step:

$$\sum_{u \in V_m} x_{u,p} \leq 1, \qquad \forall p, \tag{6}$$

$$\sum_{p \in P(u,w)} x_{u,p} = 1, \qquad \forall u. \tag{7}$$

The first set of constraints prevents overlaps of movable nodes and the second makes sure that each movable node is assigned a unique position. The delay of each edge in $E_c \cup E_s$ is determined by the location of its endpoints:

$$t_{u,v} = \sum_{p_u \in P(u,w), p_v \in P(v,w)} \tau_{p_u,p_v} e_{u,v,p_u,p_v}, \tag{8}$$

$$e_{u,v,p_u,p_v} \in \{0, 1\}, \qquad \forall p_u, p_v, \tag{9}$$

$$e_{u,v,p_u,p_v} \leq x_{u,p_u}, \tag{10}$$

$$e_{u,v,p_u,p_v} \leq x_{v,p_v}, \tag{11}$$

$$e_{u,v,p_u,p_v} + 1 \geq x_{u,p_u} + x_{v,p_v}. \tag{12}$$

Here, τ_{p_u,p_v} are constants corresponding to the least delay of the connection with its endpoints placed at p_u and p_v, respectively. Constraints (10)–(12) are merely one way of linearizing a product of two binary variables. With the timing graph modeled as in the selection LP (constraints (2)–(3)), we have a complete formulation of the placement ILP. If (u, v) is an edge from E_s, some variables become constants and the corresponding constraints simplify. This formulation is generic and can be used to place circuits on architectures with or without direct connections. It is also rather intuitive and well known in literature. In very similar form, it has been used for SAT [17] and SMT-based [16] timing-driven FPGA placement, and for ILP-based wirelength-driven ASIC placement [1].

The problem lies in the large number of position variables and quadratic length increase of delay assignment constraints (8) with respect to that number. If we fix w to 3, which is the length of the longest connection in the architecture of Figure 2, we obtain $7 \times 7 = 49$ clusters and 490 potential positions for each movable node. Any edge can have up to $490^2 > 240{,}000$ addends in the delay assignment constraint. This is clearly an issue and we address it in the next section.

B. Exploiting the Sparsity of Dedicated Interconnect

Direct connections are sparse. If they were not, the width and count of the additional multiplexers and the increased loading of LUT outputs would greatly reduce their speed, slowing down the general routing as well. In the architecture of Figure 2, there are only 14 connections originating in one cluster.

Let $E_d(u, v) \subseteq P(u, w) \times P(v, w)$ be the set of direct connections that can implement (cover) the edge $(u, v) \in E_c$. The exact position of a LUT matters only when an edge $e \in E_c$ incident to it is being covered. In all other cases, knowing its cluster is sufficient. Instead of listing all exact positions for all movable LUTs and inspecting which edges are covered, we can list the edge covering possibilities and derive the LUT positions from them. Let $C(u, w)$ be the set of clusters within the square of half-width w, centered at the initial cluster of u. A binary variable $x_{u,c}$ indicates that u is placed in cluster $c \in C(u, w)$. We can now model the edge delays as follows:

$$t_{u,v} = \sum_{(p_u,p_v) \in E_d} \tau_{p_u,p_v} e_{u,v,p_u,p_v} + nc_{u,v} \sum_{c_u \in C(u,w), c_v \in C(v,w)} \tau_{c_u,c_v} e_{u,v,c_u,c_v}, \tag{13}$$

$$nc_{u,v} = 1 - \sum_{(p_u,p_v) \in E_d} e_{u,v,p_u,p_v}. \tag{14}$$

If there is a direct connection that covers the edge (u, v) in the current subproblem, the appropriate τ from the first sum will determine the delay because the *coverage indicator* $nc_{u,v}$ will be 0. In all other cases, the indicator will be 1, causing the second sum to determine the delay. The τ constants in that sum are the delays between the two appropriate clusters, as modeled by the general placer. The e_{u,v,c_u,c_v} variables are products of the cluster position variables, linearized using constraints similar to (10)–(12). Another level of linearlization is applied to products with the coverage indicators.

The maximum length of the delay assignment constraint for $w = 3$ and the architecture of Figure 2 is now $49 \times 14 + 49^2 \ll$

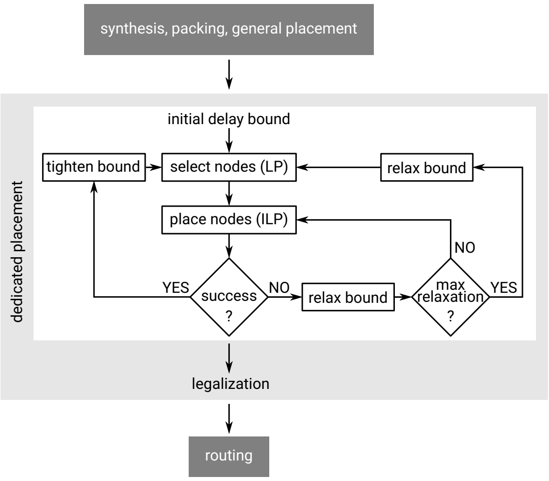

Fig. 5: The overall placement flow. Delay bounds in the loops are obtained through binary search. Once the search converges, the last successful placement is legalized.

490^2. The first addend corresponds to at most 49 ways to choose the starting cluster for the direct connection and 14 ways to choose the exact direct connection leaving it, while the second addend amounts to the number of cluster pairs that determine the edge delay if it is implemented as programmable. Similarly, the number of exact position variables for each node $u \in V_m$ is reduced from $|P(u,w)|$ to $|\cup_{\{u,v\}\in E_c}\{p_u : \{p_u, p_v\} \in E_d(u,v)\}|$; that is, to only those positions implied by covering of some edge incident to the given node.

Our model is still not complete. The linearizations (10)–(12) are of course kept, and they cause any e_{u,v,p_u,p_v} to imply exact positions of u and v. What is missing is that each exact position $x_{u,p=(x,y,i)}$ implies the corresponding cluster position $x_{u,c=(x,y)}$. The following set of constraints achieves that:

$$x_{u,c=(x,y)} \geq \sum_i x_{u,p=(x,y,i)}, \quad \forall u \in V_m, c \in C(u,w). \quad (15)$$

Finally, we need to make sure that each node is assigned exactly one cluster, using constraints similar to (7).

V. The Overall Placement Flow

The complete dedicated placement flow, along with its position within the overall CAD flow, is shown in Figure 5. It starts by computing a lower bound on the postplacement critical path delay of the given circuit. This is done by replacing the delay of each connection in the timing graph by the delay of the fastest direct connection that can implement it, given the movement restrictions of its endpoints. The bound determines the interval for the binary search represented by the outer loop of the diagram, which provides the target delay for the movable node selection LP, introduced in Section III. From the solution of the selection LP, the appropriate placement ILP is constructed, as described in Section IV. The target delay is imposed as a constraint on the maximum arrival time and the ILP is solved until the first feasible solution is found.

As discussed in Section III, the movable nodes are rarely selected in a way that actually allows for meeting the target bound. Relaxing this bound when constructing the placement ILP allows some connections selected for improvement to retain their original delay, or even increase it, thus resolving the node positioning conflicts caused by competing connection delay improvement requirements. We do not know a priori how large the minimum relaxation that allows for finding a feasible placement is, but we can impose a maximum beyond which it is likely wiser to seek a smaller set of movable nodes that can meet it. Successfully solving a placement ILP for any relaxation smaller than this maximum results in the outer binary search progressing towards larger, harder to place sets. In principle, to minimize the final delay, it suffices to minimize the relaxation of the last successfully placed set. Until this set is determined, we can fix the relaxation to its maximum value, which stands the greatest chance of resulting in a feasible problem. However, more constrained problems can in general be easier to solve and even smaller, because some position pairs can be easily filtered out during the problem construction, as they would result in the connection between the corresponding nodes being too slow. We found that performing a binary search for the minimum relaxation for each set reduces the overall solution time, both by posing better constrained placement problems and reducing the outer binary search intervals. This is represented by the inner loop of the diagram.

The last successfully solved ILP determines the placement, which may contain some overflowed clusters. We legalize them using the algorithm of Darav et al. [4], modified to price node movement in proportion to the resulting worst negative slack. VPR's net-weighting costs [15] are used to break the ties. Nodes in V_m that have a covered incident connection are kept fixed during legalization.

VI. Experimental Setup

We inherit the delay modeling from the architectural study that produced the architecture of Figure 2 [18]. We also retain the experimental methodology, along with its limitations. Notably, we do not support carry chains, fracturable LUTs, nor sparse crossbars at the moment. One important restriction of the previous methodology is now lifted, however. We extended VPR to support cluster output equivalence specification after placement, independently for each cluster. As a result, there are no longer any constraints on route-time LUT permutation for the reference architecture, while for the one with the direct connections, only those LUTs that actually use a direct connection are kept fixed; the others may be freely permuted by the router. To further improve realism, we allow each cluster output in both architectures to reach all four adjacent routing channels, similarly to the case of the Agilex architecture [2]. At the moment, we do not have a sound method for legalizing the number of inputs to each cluster, so we increase the number of physical cluster inputs to 60 for both architectures (the maximum for a ten 6-LUT cluster). As this is not uncommon in industrial architectures [7], [12], we do not believe that it has any impact on the validity of the results.

All experiments were performed on a 20-core (40-thread) Xeon-based server with 256 GB of RAM, using CPLEX 12.10 with a timeout of 10 minutes for the solver.

	postplacement							postrouting				
		lower bound		covered								
circuit	ref.	$w = 0$	$w = 1$	$w = 0$	$w = 1$	legal.	$-\Delta[\%]$	ref.	w/ dir.	mux	$-\Delta[\%]$	w/o dir.
raygentop	4.70	4.70	4.70	4.70	4.70	4.70	0.00	4.87	4.88	0.01	-0.20	4.88
ch_intrinsics	3.15	3.15	2.50	3.15	3.15	3.15	0.00	3.28	3.28	0.00	0.00	3.28
mkDelayWorker32B	6.83	6.79	6.55	6.83	6.58	6.58	3.66	7.09	7.09	0.06	0.00	7.36
bgm	23.56	23.56	22.21	23.56	23.33	23.33	0.98	23.66	23.60	0.14	0.25	25.59
mkSMAdapter4B	5.16	5.11	4.72	5.11	5.02	5.02	2.71	5.38	5.32	0.03	1.11	5.82
stereovision0	3.74	3.74	3.31	3.74	3.59	3.59	4.01	3.74	3.60	0.01	3.74	4.00
boundtop	6.10	6.01	5.57	6.01	5.73	5.73	6.06	6.05	5.78	0.08	4.46	6.18
diffeq1	20.45	19.47	17.78	20.45	19.22	19.22	6.01	21.16	20.01	0.09	5.43	21.69
diffeq2	15.69	14.92	12.80	15.06	14.65	14.65	6.63	16.14	15.20	0.11	5.82	16.74
LU8PEEng	105.05	101.07	91.47	103.85	97.10	97.10	7.57	104.86	97.57	0.95	6.95	109.20
blob_merge	9.90	8.76	6.79	9.49	8.85	9.17	7.37	9.89	9.18	0.03	7.18	10.14
or1200	13.08	12.66	10.70	12.81	11.73	11.80	9.79	13.12	12.09	0.24	7.85	15.69
sha	11.89	11.02	9.15	11.44	10.74	10.88	8.49	11.88	10.94	0.09	7.91	12.53
geomean	10.29	9.98	8.85	10.15	9.75	9.79	4.86	10.46	10.05		3.92	10.99

TABLE I: Numerical results. All delays are in nanoseconds.

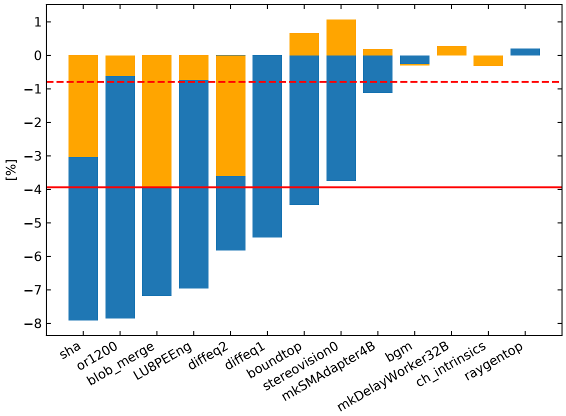

Fig. 6: Relative change in the postrouting critical path delay. The $w = 0$ and the $w = 1$ cases are shown in orange and blue, respectively. The dashed red line represents the relative change of the geometric mean critical path delay over all circuits, for $w = 0$, while the solid line represents the same for $w = 1$.

VII. Results

In this section, we report the results of applying the previously described algorithms. All results are medians of five different placements and each circuit was routed by the *delay targeted* routing algorithm of Rubin and DeHon [19], implemented on top of VTR 7.0 [13], with the channel width fixed at 300 tracks. The maximum delay bound relaxation used when constructing the placement ILPs was set to 10%.

A. Delays

Postplacement delays for the cases when the LUTs are allowed to move only within their initial clusters ($w = 0$) and in a region of 3×3 clusters centered at the initial clusters ($w = 1$) are shown in the columns labeled as *covered* in Table I. The 400 ps difference between the average delays of 10.15 ns and 9.75 ns is significant and translates to almost $4\times$ greater relative average improvement over the reference, when LUTs are allowed to change clusters.

We may note that the 1.36% average improvement for the $w = 0$ case is much lower than what was reported in prior work [18]. This could be an artifact of an inferior movable node selection method, although the lower bounds in Table I suggest a more fundamental cause. The cause is in fact of architectural nature: because we use a 60- instead of a 40-input cluster architecture, a much denser packing is obtained, bringing some of the intercluster routing delay into the clusters. Since the architecture has no local direct connections, when $w = 0$, the placer cannot do anything to improve their delay, while when $w > 0$ it can. This is visible in Figure 1 as well. To verify the hypothesis, we reran the experiments for $w = 0$ on a subset of circuits for which the average relative improvement was 1.77%, using a 40-input architecture. The improvement rose to 3.21%, which is much closer to the previously reported results [18]. This illustrates how important it is to consider other architectural parameters when deciding which direct connections are the most beneficial.

The placements for $w = 0$ are legal by construction, but those for $w = 1$ are not. The postlegalization results are also reported in Table I. The delay does sometimes deteriorate due to legalization, but in most cases by a modest amount.

Finally, the postrouting delays are reported in the column designated as *w/ dir*. The postlegalization relative improvement is generally retained throughout the routing process. Many of the cases where a nonnegligible deterioration occurs can be explained by the delays of the additional multiplexers that are not modeled during placement. Those circuit connections that are implemented as direct are forced to suffer this additional delay, while the others are rarely impacted by it. This is due to the sparsity of the direct connections, which causes relatively few LUT inputs to be delayed. The difference that dedicated placement brings to the postrouting delay is shown in Figure 6.

In the placement ILP formulation, we allow connection delays to decrease only when implemented as direct. However, it is possible that some of the delay improvements in Table I are due to shortening of programmable connections or implicit packing improvement. To verify if this is the case, we repeated the experiments, routing the circuits placed with the dedicated algorithm, without actually using the direct connections. The results are reported in the *w/o dir.* column of Table I. Clearly, it is not the overall improvement of placement and packing that lead to the positive results. In fact, the dedicated placer significantly distorts the general placement, in a way that makes sense only in the presence of direct connections.

	size				components			degrees				
circuit	$\lvert V\rvert$	$\lvert E\rvert$	W	H	#	$\langle\lvert V\rvert\rangle$	max$\lvert V\rvert$	⟨total⟩	max total	max in	max out	longest path
stereovision0	15	10	6	7	5	3.00	4	1.33	3	3	2	2
	13	8	6	7	5	2.60	3	1.23	2	2	2	2
boundtop	40	24	13	13	16	2.50	5	1.20	3	2	2	4
	32	16	13	12	16	2.00	2	1.00	1	1	1	2
diffeq1	30	24	7	6	6	5.00	12	1.60	3	1	2	6
	28	18	7	6	10	2.80	4	1.29	2	1	2	4
diffeq2	32	24	6	4	8	4.00	10	1.50	4	3	2	6
	28	16	6	4	12	2.33	3	1.14	2	1	1	3
LU8PEEng	252	161	23	24	91	2.77	7	1.28	4	3	2	6
	210	117	23	24	93	2.26	4	1.11	2	2	2	3
blob_merge	43	35	4	7	9	4.78	18	1.63	5	2	4	9
	38	22	4	7	16	2.38	4	1.16	3	2	3	3
or1200	81	62	16	13	19	4.26	13	1.53	2	1	1	13
	73	40	16	13	33	2.21	3	1.10	2	1	1	3
sha	58	43	13	8	16	3.62	13	1.48	4	2	2	7
	48	27	13	8	21	2.29	3	1.12	2	2	2	3

TABLE II: Properties of the subgraphs induced by the connections selected for improvement. The shaded rows show the corresponding properties of the subgraphs induced by the connections that were successfully improved by being implemented as direct. Columns W and H correspond respectively to the width and the height, in number of clusters, of the region bounding the movable nodes.

B. Improvement Subgraphs

The size and the structure of the circuit subgraphs induced by the connections selected for improvement (the solid edges in Figure 7) influence both the time needed to solve the placement ILPs and the achievable critical path delay reduction. Some basic properties of the last successfully placed subgraph in the run resulting in the median postplacement delay are given in Table II. The circuits that achieved a final delay improvement of $< 3\%$ are omitted, as their subgraphs were either very small, or no successful placement was found for any of them.

Perhaps the most apparent feature of the subgraphs is their fragmentedness, visible in the *components* columns which show the sizes of the *weakly connected components* (maximal subgraphs where every node can be reached from all others by paths that may be traversed in reverse). The longest path often retains substantial length, however. The node degrees are low, which is appropriate for the architecture of Figure 2.

The subgraphs induced by the connections that are actually implemented as direct (the blue edges in Figure 7) are noticeably smaller than the ones originally selected for improvement, but they still cover a large portion of their edges.

Without the information on how the individual connections selected for improvement are positioned within the entire circuit graph, it is not apparent how covering each of them influences the reduction of the critical path delay. We show one particular improvement subgraph in Figure 7. The dashed arrows mark the edges between the movable nodes that were not selected for improvement. It should not be surprising that they often occur as intermediate edges of paths that were selected for improvement. The intention of the selection LPs of Section III—although there is no guarantee that it will actually be realized—is to select a minimal subset of edges of a path as this directly influences the size of the placement ILPs. We can see that, in this case, the numerous small connected components are not merely pieces of unrelated paths, but in fact constitute a carefully selected subgraph of a nontrivial graph. This showcases the generality of the movable node selection method that was mentioned in Section II-A.

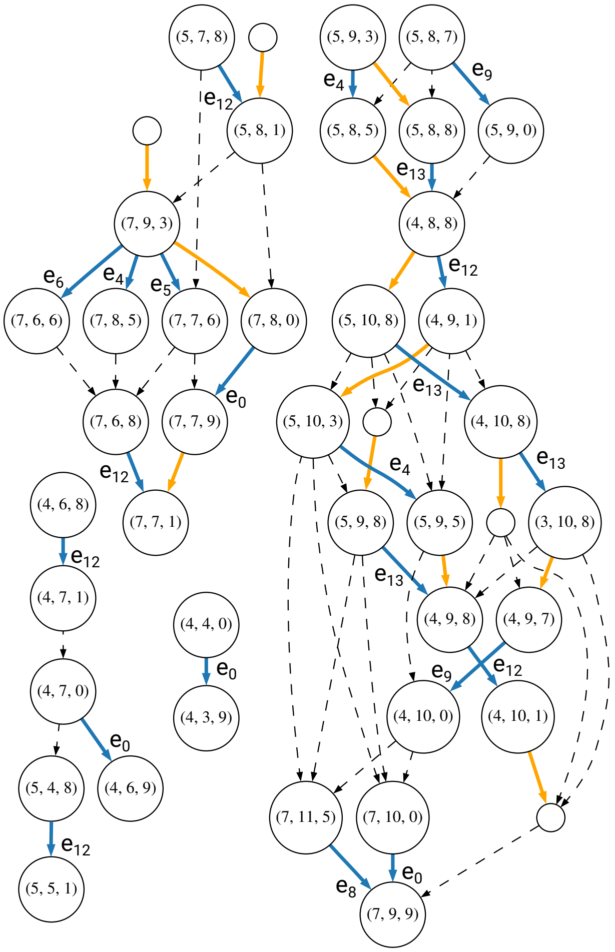

Fig. 7: Subgraph selected for improvement in the *blob_merge* circuit. All nodes are movable. The blue solid arrows correspond to the edges that were successfully implemented as direct after dedicated placement. The orange solid arrows represent the edges selected for improvement that were not successfully implemented as direct. Finally, the dashed arrows depict the edges that exist between the movable nodes but were not selected for improvement. Those nodes that have an incident direct connection have their final positions as labels. The label of each blue edge corresponds to the identifier used in Figure 2 to mark the direct connection that implemented it.

		problems				status			flexibility (last feas.)			⟨time⟩	
circuit	#LUTs	LP	⟨sol. t⟩	ILP	⟨setup t⟩	feas.	infeas.	timeout	⟨pos./u⟩	⟨pair/e⟩	⟨|tree|⟩	feas.	infeas.
diffeq2	322	4	0.02	7	1.51	2	5	0	39	78	54	2.06	0.50
ch_intrinsics	425	3	0.01	3	0.38	0	3	0	-	-	0	-	0.17
diffeq1	485	6	0.04	6	1.81	6	0	0	39	72	167	1.79	-
mkSMAdapter4B	1 982	3	0.11	5	0.67	2	3	0	26	32	0	0.02	0.02
sha	2 280	2	0.21	9	4.62	3	3	3	39	88	12 848	12.00	65.31
raygentop	2 429	0	-	0	-	0	0	0	-	-	-	-	-
or1200	3 054	5	0.29	18	3.55	8	10	0	43	76	375	6.04	9.47
boundtop	3 070	3	0.20	10	3.29	5	5	0	30	58	0	0.52	0.02
mkDelayWorker32B	5 602	3	0.32	7	1.85	3	4	0	38	39	0	0.03	0.03
blob_merge	6 019	4	0.46	11	2.45	4	7	0	44	94	1 647	25.57	21.01
stereovision0	14 779	3	0.54	10	3.33	5	5	0	29	51	0	0.08	0.03
LU8PEEng	26 455	3	6.36	10	45.17	5	5	0	35	80	10 754	9.15	36.81
bgm	36 480	1	7.57	3	10.62	0	3	0	-	-	0	-	0.01

TABLE III: Solution runtime. All runtimes are in seconds.

Another interesting observation that can be made from Figure 7 is that the connections that were successfully covered use a wide variety of direct connections available in the architecture, with different span lengths and directions, both vertical and horizontal. This seems to confirm what was found in prior work [18]: that using only very simple patterns of direct connections, such as the vertical cascades, may not expose their full potential.

C. Runtimes

Runtime breakdowns for the placement run that resulted in the median postplacement delay of the given circuit are reported in Table III. In most cases, the number of selection LPs solved to determine the movable node sets is fairly low. The average time needed to solve them is almost negligible, but there is an obvious increasing trend corresponding to the circuit size, which is expected. This solution time can be further reduced by considering only the critical subgraphs of the timing graphs.

The number of solved placement ILPs is also fairly low. The infeasible ILPs occur slightly more often than the feasible ones, while those that time out are rare. There seems to be no correlation between the size of the circuit and the solution time, which is expected, as the size of the movable node set has no a priori correlation with the circuit size either. Some of the ILPs are solved by merely solving the continuous relaxation of the problem in the root of the search tree ($\langle|\text{tree}|\rangle = 0$). Others, however, require substantial branching. In these cases, the capability of CPLEX to branch in parallel is very useful. For this size of the search trees, memory is, however, not a concern. The largest trees required on the order of a few hundreds of megabytes. Each ILP also needs time to be constructed. This time is reported under the ⟨*setup t*⟩ column. In some cases, it is nontrivial, but this is mostly due to a fairly inefficient Python implementation.

The average number of positions per movable node (⟨pos./u⟩) is substantial in most cases. It is lower than the theoretical maximum of 90, however. The average number of covering pairs for each edge (⟨pair/e⟩) is also much lower than the 8100 that would occur in the naive formulation.

All experiments so far were performed for $w \leq 1$. Further increasing the freedom of movement results in placement ILPs that are too large to be solved within a reasonable time limit without an incremental approach [11]. For $w = 3$, the number of variables and constraints becomes so large that even a solution to the continuous relaxation of the root problem cannot be obtained within 10 minutes. To assess the possibilities for further delay improvement that increasing the freedom of movement could bring, we ran experiments with $w = 2$, on *blob_merge*, *or1200*, and *sha* circuits, fixing the placement ILP target to 200 ps less than the incumbent solution for $w = 1$. The corresponding selection LP target was fixed to yet 700 ps less, to allow for ample relaxation. A feasible solution was found for each circuit, in 707, 728, and 1034 seconds, respectively. This demonstrates that increased freedom of movement can bring further delay reductions, amounting to at least 200 ps, which represents about 17% of the average delay reduction achieved for these three circuits with $w = 1$.

VIII. Conclusions and Future Work

In this work, we introduced an efficient ILP-based placement algorithm for FPGA architectures with direct connections between LUTs, which vastly improves their effectiveness compared to architecture-oblivious algorithms. We also removed some important limitations of the previously used experimental methodology and showed that the direct connections continue to bring benefits in this more realistic setting.

The fact that a simple change in the underlying architecture—increase in the number of cluster inputs—substantially altered the conclusions about the utility of a particular type of direct connections suggests that a comprehensive study of the mutual influence of direct connections and other architectural parameters is due. Dedicated placement could also have a strong influence on architectural decisions. Hence, extending the algorithm presented here with support for classical architectural features required by such a study is of great importance.

Our experiments showed that increasing the freedom of movement beyond what was done in this work would lead to increased benefits. Another future step on the algorithmic front should therefore be to address the scalability issues that prevent this at the moment, by integrating incremental solution approaches, or even other solution techniques, such as SAT, that could be better suited to the nature of the problem.

An implementation of the placement algorithm presented in this paper can be found at https://github.com/stefannikolicns/fpl20-placement.

References

[1] S. Cauley, V. Balakrishnan, Y. C. Hu, and C. Koh. A parallel branch-and-cut approach for detailed placement. *ACM Trans. Design Autom. Electr. Syst.*, 16(2):18:1–18:19, 2011.

[2] J. Chromczak, M. Wheeler, C. Chiasson, D. How, M. Langhammer, T. Vanderhoek, G. Zgheib, and I. Ganusov. Architectural enhancements in Intel® Agilex™ FPGAs. In *FPGA '20: The 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Seaside, CA, USA, February 23-25, 2020*, pages 140–149. ACM, 2020.

[3] K. C. K. Chung. *Architecture and Synthesis of Field-Programmable Gate Arrays with Hard-wired Connections*. Ph.D. thesis, University of Toronto, Toronto, 1994.

[4] N. K. Darav, A. A. Kennings, K. Vorwerk, and A. Kundu. Multi-commodity flow-based spreading in a commercial analytic placer. In *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, FPGA 2019, Seaside, CA, USA, February 24-26, 2019*, pages 122–131. ACM, 2019.

[5] S. Dhar, S. N. Adya, L. Singhal, M. A. Iyer, and D. Z. Pan. Detailed placement for modern FPGAs using 2D dynamic programming. In *Proceedings of the 35th International Conference on Computer-Aided Design, ICCAD 2016, Austin, TX, USA, November 7-10, 2016*, pages 1–8. ACM, 2016.

[6] S. Dhar, M. A. Iyer, S. N. Adya, L. Singhal, N. Rubanov, and D. Z. Pan. An effective timing-driven detailed placement algorithm for FPGAs. In *Proceedings of the 2017 ACM on International Symposium on Physical Design, ISDP 2017, Portland, OR, USA, March 19-22, 2017*, pages 151–157. ACM, 2017.

[7] W. Feng. K-way partitioning based packing for FPGA logic blocks without input bandwidth constraint. In *2012 International Conference on Field-Programmable Technology, FPT 2012, Seoul, Korea (South), December 10-12, 2012*, pages 8–15. IEEE, 2012.

[8] S. E. Hambrusch and H.-Y. Tu. Edge weight reduction problems in directed acyclic graphs. *Journal of Algorithms*, 24(1):66 – 93, 1997.

[9] M. D. Hutton, V. Chan, P. Kazarian, V. Maruri, T. Ngai, J. Park, R. H. Patel, B. Pedersen, J. Schleicher, and S. Shumarayev. Interconnect enhancements for a high-speed PLD architecture. In *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays, FPGA 2002, Monterey, CA, USA, February 24-26, 2002*, pages 3–10, 2002.

[10] C. Li, M. Xie, C. Koh, J. Cong, and P. H. Madden. Routability-driven placement and white space allocation. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 26(5):858–871, 2007.

[11] S. Li and C. Koh. Mixed integer programming models for detailed placement. In *International Symposium on Physical Design, ISPD'12, Napa, CA, USA, March 25-28, 2012*, pages 87–94. ACM, 2012.

[12] W. Li and D. Z. Pan. A new paradigm for FPGA placement without explicit packing. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 38(11):2113–2126, 2019.

[13] J. Luu, J. Goeders, M. Wainberg, A. Somerville, T. Yu, K. Nasartschuk, M. Nasr, S. Wang, T. Liu, N. Ahmed, K. B. Kent, J. Anderson, J. Rose, and V. Betz. VTR 7.0: Next generation architecture and CAD system for FPGAs. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 7(2):6:1–6:30, June 2014.

[14] I. L. Markov, J. Hu, and M. Kim. Progress and challenges in VLSI placement research. In *2012 IEEE/ACM International Conference on Computer-Aided Design, ICCAD 2012, San Jose, CA, USA, November 5-8, 2012*, pages 275–282. ACM, 2012.

[15] A. Marquardt, V. Betz, and J. Rose. Timing-driven placement for FPGAs. In *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays, FPGA 2000, Monterey, CA, USA, February 10-11, 2000*, pages 203–213. ACM, 2000.

[16] A. Mihal. A difference logic formulation and SMT solver for timing-driven placement. In *SMT Workshop 2013 11th International Workshop on Satisfiability Modulo Theories, Helsinki, Finland, July 8-9, 2013. Informal Proceedings*, pages 16–25, 2013.

[17] A. Mihal and S. Teig. A constraint satisfaction approach for programmable logic detailed placement. In *Theory and Applications of Satisfiability Testing - SAT 2013 - 16th International Conference, Helsinki, Finland, July 8-12, 2013. Proceedings*, volume 7962 of *Lecture Notes in Computer Science*, pages 208–223. Springer, 2013.

[18] S. Nikolić, G. Zgheib, and P. Ienne. Straight to the point: Intra- and intercluster LUT connections to mitigate the delay of programmable routing. In *FPGA '20: The 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Seaside, CA, USA, February 23-25, 2020*, pages 150–160, Seaside, CA, USA, February 2020.

[19] R. Rubin and A. DeHon. Timing-driven pathfinder pathology and remediation: quantifying and reducing delay noise in VPR-pathfinder. In *Proceedings of the ACM/SIGDA 19th International Symposium on Field Programmable Gate Arrays, FPGA 2011, Monterey, California, USA, February 27, March 1, 2011*, pages 173–176. ACM, 2011.