2026 IEEE 34th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)

Déjà Vu Packing: Optimizing FPGA Logic Clustering Runtime via Pattern Memoization

Milo Liebster[†], Amin Mohaghegh[*], Andrew Boutros[†]
[†]Department of Electrical and Computer Engineering, University of Waterloo
[*]QuickLogic Corporation
Email: {pliebste, andrew.boutros}@uwaterloo.ca, amohaghegh@quicklogic.com

Abstract*—Implementing a digital circuit on a field-programmable gate array (FPGA) fabric requires clustering technology-mapped netlist primitives into coarser-granularity blocks that can be directly mapped to the physical resources available on the FPGA fabric. As the internal architecture of FPGA logic blocks (LBs) has grown in complexity, with sophisticated logic elements (LEs) and highly irregular local interconnect, this packing problem has become significantly more challenging. To ensure the feasibility of intracluster routing, the computer-aided design (CAD) tools must solve a costly multi-source multi-sink routing problem for each candidate cluster. In this paper, we first show that such packing legality checks consume a significant portion of the CAD flow runtime for LB architectures with complex LEs and local routing structures resembling modern commercial FPGAs. We demonstrate that the packing stage constitutes 58% and 94% of the entire Versatile Place and Route (VPR) flow runtime on average when mapping a wide variety of benchmarks to the AMD 7-series-like and Altera Stratix 10–like VTR architecture captures, respectively. By analyzing the packing algorithm behavior, we observe that a significant fraction of the attempted packed clusters are repetitions of a much smaller number of packing patterns, and therefore many of the packing legality checks are redundant and could be skipped. To this end, we introduce our Déjà Vu packing approach, which leverages a novel packing signature tree data structure that enables efficient identification of recurring packing patterns and memoization of their legality check outcomes. Our approach speeds up the packing runtime by up to 13.4× and 29.3×, with an average of 3.7× and 6.9×, across the evaluated benchmarks on the 7-series and Stratix 10 architecture captures. These packing runtime gains result in a significant 1.6× and 5.3× average reduction in end-to-end VPR runtime, while maintaining quality of results.

I. Introduction

Field-programmable gate array (FPGA) development relies on a multi-stage computer-aided design (CAD) flow that maps descriptions of digital circuits onto the configurable fabric. Synthesis and technology mapping first translate the design logic into a netlist of *primitives* available on the FPGA, such as lookup tables (LUTs), flip-flops (FFs), adders, multipliers, and memory elements. Then, an explicit packing stage can be used to organize these primitives, based on their types, into coarser-granularity *clusters* that can be mapped to the *physical blocks* in the target FPGA architecture, such as logic blocks (LBs), digital signal processing blocks, and block RAMs. Poor packing decisions at this step directly impact downstream placement and routing quality, resulting in increased wire congestion, longer critical paths, and reduced overall circuit performance. Alternatively, some contemporary CAD flows first perform an initial placement of netlist primitives to guide subsequent packing decisions or directly place netlist primitives at specific cluster locations and iteratively refine any invalid or sub-optimal cluster mappings [1].

Among the various block types in modern FPGAs, LBs are the most challenging to cluster due to their architectural complexity and heterogeneity. They contain tens of LUTs, FFs, and hardened adders [2], all interconnected through a complex and highly irregular local routing network. Unlike simpler block types with tighter constraints, modern LB architectures offer numerous possible configurations for clustering primitives, but not all clusters are physically realizable (i.e., *legal*). The CAD flow must ensure that each candidate cluster has a valid intracluster routing solution. This involves costly legality checks that require solving multi-source, multi-sink routing problems within the constrained local interconnect graph. Since many candidate configurations may be attempted per cluster (in an explicit packing stage or during flat placement legalization), these legality checks are performed thousands or even millions of times, consuming substantial runtime.

By analyzing the runtime of the widely used open-source Versatile Place and Route (VPR) [3] tool, we demonstrate that cluster legality checks during the packing stage constitute the majority of the flow runtime for architectures that resemble modern commercial FPGAs with complex LBs. However, our analysis also reveals that the runtime spent checking cluster legality is dominated by redundant checks, assessing the legality of recurring cluster patterns that have been previously encountered and evaluated during packing. Based on this observation, we present a novel *packing signature tree* (PST) data structure that enables efficient memoization and comparison of cluster packing patterns by assigning them unique representations encoded as paths through a tree, which we refer to as *packing signatures*. Then, we integrate our PST into VPR's packing algorithm to characterize a wide variety of circuits from the Verilog-to-Routing (VTR), Koios [4], Titan23 [5], and Titanium25 [3] benchmark suites. Our experiments target both the AMD 7-series and Altera Stratix 10 VTR architecture captures. However, we modify the VTR Stratix 10 capture to more accurately model the complex LB internals of the commercial Stratix 10 device.

We find that, on average, 62% of the clusters subjected to legality checks are repeats of clusters that have been previously evaluated for legality. We show that by memoizing the legality check outcomes of these clusters within our PST, enabling redundant checks to be skipped, the VPR packing runtime can be reduced by up to 29.3×. This approach speeds up the entire VPR flow by an average of 1.6× and 5.3× across 68 benchmarks on the 7-series and Stratix 10 architectures, respectively. Our proposed memoization strategy does not involve any heuristics; therefore, it results in identical packing solutions, and thus quality of results (i.e., routed wirelength and critical path delay), to those produced by the unmodified

2576-2621/26/$31.00 ©2026 IEEE
DOI 10.1109/FCCM68464.2026.00016

flow. In summary, our contributions include:

- Demonstrating that the majority of clusters produced by greedy seed-based packing recur from a much smaller set of distinct packing patterns.
- Introducing a novel packing signature tree data structure to memoize and enable efficient recognition of recurrent packing patterns and their legality check outcomes.
- Developing a Stratix 10–like VTR architecture file that faithfully captures the complexity of the commercial Stratix 10 LB architecture.
- Reducing VPR's packing and end-to-end runtime by 5.1× and 2.9× on average across a wide variety of benchmarks and two architecture captures of modern commercial devices with complex LBs.

Our packing memoization technique is integrated into VTR and can be configured using the command-line argument `--memoize_cluster_packings {on|off}`.

II. Background & Related Work

A. FPGA Packing

Most of the prior work on FPGA packing algorithms focuses on seed-based approaches in which the packer iteratively selects a *seed* primitive to form a new cluster and then greedily adds related primitives until the cluster is full or no more legal additions can be made. One of the earliest FPGA seed-based packers is VPack [6], which aims to maximize cluster utilization through simple heuristics. The unpacked primitive with the most used pins is selected as a cluster seed, and subsequent primitives are greedily added to the cluster based on their connectivity to already-packed elements. T-VPack [7] extends VPack by making it timing-driven. It successively clusters primitives along the circuit's critical path to minimize the number of external connections between them and re-evaluates primitive criticality after each decision.

Other seed-based packers explore alternative optimization objectives and clustering strategies. For example, RPack [8] uses a packing cost function that aims to improve routability, while iRAC [9] improves routing congestion by trying to match the Rent exponent of the clustered design to that of the underlying FPGA architecture. HD-Pack [10] performs a rapid global placement to determine approximate primitive locations and uses this information to better inform packing decisions. MO-Pack [11] integrates ideas from multiple prior works to co-optimize energy, critical path delay, and channel width concurrently using a many-objective packing algorithm. Depopulation-based packers, such as Un/DoPack [12] and T-NDPack [13], improve routing congestion by deliberately producing partially filled clusters. This technique spreads out the circuit across a larger area of the fabric and therefore reduces the peak utilization of routing channels.

Besides seed-based approaches, several prior works explore partitioning-based packing algorithms that split the netlist into an initial set of clusters using a k-way partitioner, such as hMetis [14], and then perform a legalization step to fix these clusters. PPack [15], [16] uses recursive bipartitioning to optimize the Rent characteristic of the cluster-level netlist instead of simply minimizing the number of external signals. More recent partitioning approaches such as PartSA [17] and MultiPart [18] have incorporated multithreading to improve packing runtime. Singhal et al. [19] present a consensus-based packing technique in which each primitive independently creates candidate clusters and iteratively reaches consensus with other members of the cluster. This approach mitigates the greedy behavior of seed-based packers while enabling multithreaded clustering since many primitives can concurrently form their candidate clusters and evaluate their legality. Since accurate and fast estimates of timing, wirelength, and congestion of the placed netlist are not available during the packing stage, some prior work [20], [21] allow packed primitives to retroactively be moved and swapped between clusters during placement to further improve the final implementation quality of results.

These partitioning-based and hybrid methods have not achieved widespread adoption in academic tools, largely due to their reduced adaptability across different FPGA architectures and heterogeneous blocks compared to seed-based approaches. The current academic state-of-the-art for seed-based packing is Architecture-Aware Packing or AAPack [22], [23] developed for the VTR [3] open-source FPGA toolflow. VTR takes as an input an XML-based file that describes the details of an FPGA architecture, such as its block types, organization, timing/area models, and routing resources between and within blocks. The flow consists of multiple tools that synthesize benchmark circuits, map them to the specified FPGA architecture, and report implementation results such as resource utilization, critical path delay, and routing wirelength.

To enable flexible architecture exploration in VTR, AAPack can target any arbitrary logic block architecture defined using VTR's architecture description format with features such as fracturable LUTs, hardened arithmetic, and depopulated crossbars [24]. Supporting complex and arbitrary architectures has significantly increased the computational cost of cluster legality checking, as verifying routing feasibility can require solving numerous multi-source, multi-sink intracluster routing problems to form even a single cluster. AAPack introduced speculative packing to mitigate this overhead by optimistically skipping all intermediate legality checks in its first attempt to pack a cluster; however, as LB complexity increases, the savings achieved by this approach are reduced since the probability of packing a legal cluster without intermediate legality checks becomes significantly lower. To further address packing runtime, RSVPack pre-computes routing feasibility tables representing all legal cluster routing configurations of a Virtex 6 LB [25]. The table is efficiently iterated through to assess cluster legality, achieving an average 25× speedup compared to AAPack. However, it is not described whether RSVPack can evaluate a solution space involving different permutations of logically equivalent pins (i.e. interchangable cluster and LUT inputs connected by a crossbar), thus it is unclear if this method could be generalized for arbitrary architectures. To provide a general method for reducing packing runtime, our work introduces a lazy method for memoizing the results of the legality checks of previously encountered packing patterns. This eliminates redundant legality checks resulting in significant runtime gains, especially for architectures with complex LB structures. Although we showcase our proposed technique for the seed-based packing algorithm in VPR, we believe it is also applicable to other packing algorithms and CAD flows in which intracluster routing (or other costly checks) must be performed to guarantee legality.

B. Architecture Aware Packing Algorithm in VPR

Algorithm 1 summarizes the packing algorithm currently used in VPR, which was originally introduced in [22] and then

Algorithm 1: Overview of the VPR packing algorithm.

```
Function Pack(netlist) is
    while netlist.has_unpacked_molecules() do
        seed_molecule ← netlist.get_next_seed();
        cluster ← start_cluster(seed_molecule);
        /* Speculative Packing                    */
        while cluster.capacity() < max_util do
            molecule ← netlist.get_next_mol(cluster);
            if cluster.check_pin_count(molecule) then
                cluster.insert(molecule);
        if cluster.check_routability() then
            cluster.finalize();
            continue;
        /* Detailed Packing                       */
        cluster.reset();
        while cluster.capacity() < max_util do
            molecule ← netlist.get_next_mol(cluster);
            if cluster.check_pin_count(molecule) then
                cluster.insert(molecule);
                if !cluster.check_routability() then
                    cluster.remove(molecule);
                    continue;
        cluster.finalize();
```

refined over several VTR releases [3], [26], [27] to improve its runtime and quality of results (QoR). The input to the packing stage is a technology-mapped netlist of interconnected primitives, such as lookup tables (LUTs), flip-flops (FFs), memories, adders, and multipliers. First, a pre-packing step is performed which groups adjacent netlist primitives (also called *atoms*) that have highly-constrained placement relationships relative to each other into *molecules* whose atoms must be placed together in fixed arrangements during packing. Examples of such molecules are LUTs directly driving FFs, or adders connected via dedicated carry chains [26]. Primitives that do not have such constraints are pre-packed into single-atom molecules.

The packing algorithm begins by picking a seed molecule according to a gain function parameterized by pin utilization and timing criticality and placing it into a new cluster of the appropriate type (e.g., LB, BRAM, DSP) to accommodate it. Following seed placement, the algorithm continues to greedily select the most-related unpacked molecules that can be inserted into the open cluster, based on an empirically derived *attraction function*, until the user-recommended pin utilization is achieved or all candidate molecules have been exhausted. The default VPR packer settings set the target input pin utilization for LBs to 80% and disable packing of unrelated molecules, which was shown to achieve better QoR [27].

Since VTR allows the description of arbitrary block primitives and local interconnect structures, it is necessary to ensure that the intracluster routing among the molecules in a packed cluster and its external pins is feasible. The VPR packer first performs *speculative packing* in which only a simple pin counting legality check is performed each time a molecule is added to a cluster to rapidly discard packing solutions that exceed the number of available cluster pins. Then, only when a cluster is complete, the intracluster router is invoked to check its routing feasibility. If the speculative packing attempt fails the routing legality check, the cluster is reset and a *detailed packing* is performed where the intracluster router is invoked after the insertion of each molecule. This continues until the cluster is filled to the desired packing density or until the cluster becomes unroutable if any of the remaining unpacked molecules is inserted.

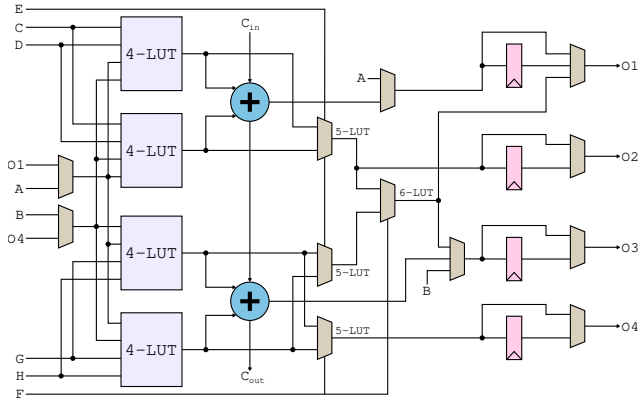

Fig. 1. Complex commercial Stratix 10 LE with 8 distinct inputs (A-H), one fracturable 6-LUT, two bits of arithmetic, and four bypassable FFs. Finding a packing solution with feasible routing can be computationally expensive for this LB organization since not all inputs can feed both 5-LUTs in fractured mode, only 2 registers (O1, O3) can be directly reached from 2 specific inputs (A, B), and adder outputs can only feed specific registers (O1, O3).

C. 7-series & Stratix 10 VTR Architecture Captures

The latest VTR release [3] includes approximate models of the AMD 7-series and the Altera Stratix 10 commercial architectures. The LB in the 7-series architecture has 56 inputs and 24 outputs split between two slices. A slice contains four logic elements (LEs), each of which has a six-input fracturable LUT (6-FLUT), two bypassable FFs, and one bit of arithmetic. The local interconnect drives each slice input from one LB input or one of three slice outputs (i.e., feedback connections), one or two of which are produced by the same slice. The two FFs in an LE can be driven by the 6-FLUT or directly reached through a specific LE input, skipping the LUTs. Additionally, one of the two FFs can be driven by the output of the hard arithmetic circuitry.

Talaei et al. [28] introduced the VTR Stratix 10 architecture capture along with an updated Titan flow [5] to support newer Altera FPGA families. The Titan flow enables synthesizing benchmarks using the Altera Quartus CAD tool to produce circuit netlists in `blif` format, which can then be packed, placed, and routed using VPR. In this architecture capture, 10 LEs are grouped into an LB with a total of 60 inputs and a full local crossbar. Each LE has eight distinct inputs that can drive *any* of the six inputs of an FLUT-6, two bypassable FFs that can optionally be directly driven by *any* LE input (skipping the LUTs), and two bits of arithmetic. This is an overly simplified model compared to the commercial Stratix 10 LE illustrated in Fig. 1. A faithful model of the Stratix 10 architecture would instead have a 50% depopulated crossbar, a more restrictive connectivity pattern between the eight LE inputs and six FLUT inputs, and four FFs per LE (two of which can be directly reached each by a single LE input,

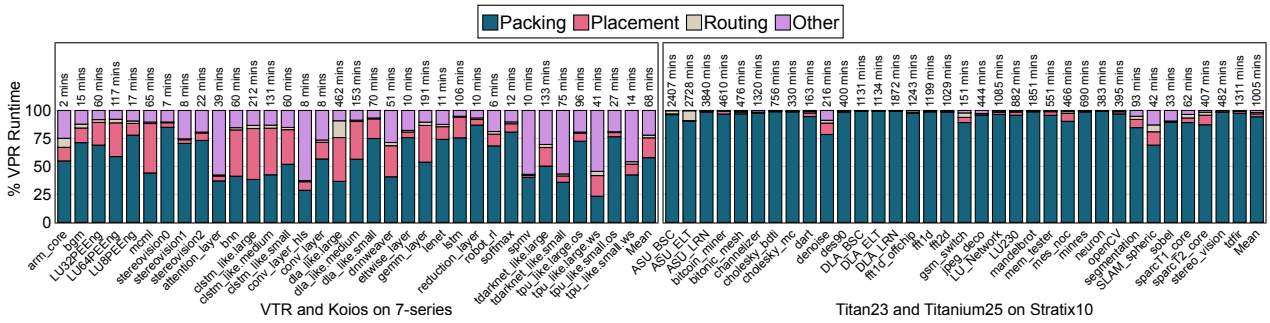

Fig. 2. VPR runtime breakdown for the VTR and Koios benchmarks on the 7-series architecture (W=300) (left), as well as the Titan23 and Titanium25 benchmarks on Stratix 10 architecture (W=400) (right). On average, the packing runtime is 58% (39 mins) and 94% (945 mins) of the overall VPR runtime (68 and 1005 mins).

skipping the LUTs). Based on communication with the authors of [28], these complex details of the Stratix 10 LB architecture were omitted as they resulted in a significant increase in the VPR packing runtime, prohibiting effective architecture and CAD algorithm evaluations. Our work solves this issue and thus enables exploration of complex LB architectures similar to those in modern commercial FPGAs.

III. Baseline Packing Results

A. Faithful Stratix 10 LB Architecture Model

To characterize the VPR packing algorithm on realistic LB architectures representative of commercial devices, we first modify the original VTR Stratix 10 architecture capture [28] to incorporate the omitted details mentioned in Section II-C. We incorporate a 50% depopulated local crossbar built out of smaller full crossbars. The LB inputs are divided into four groups of 15 logically equivalent pins where each of the eight inputs to an LE can be driven by any signal from two of the four groups. We also increase the number of FFs from two to four, and capture the details of the complex connectivity between LE primitives as illustrated in Fig. 1. Our modified architecture file has been contributed to the open-source VTR repository and will be referred to as the Stratix 10 architecture throughout the experiments described in the rest of this paper.

B. VPR Packing Characterization

In this subsection, we characterize the behavior and runtime of the VPR packer for circuits with more than $10,000$ netlist primitives from the VTR and Koios [4] benchmark suites mapped to the 7-series architecture as well as the Titan23 [5] and Titanium25 [3] benchmark suites mapped to the Stratix 10 architecture[1]. We add telemetry code to analyze the main contributors to packing runtime in the VTR master branch code at the time we started this work (commit ID: `7c0ee8a6`). All the reported results are averages of running VTR with 3 seeds on an Intel Xeon w9-3495 CPU with 1 TB of RAM.

Fig. 2 presents a breakdown of the end-to-end VPR runtime for different benchmarks and architectures. The 7-series results show that the packing stage constitutes 58% (about 39 mins) of the entire flow runtime on average. The packing runtime increases to 94% (945 mins) of the VPR runtime when mapping the Titan23 and Titanium25 benchmarks on the more complex Stratix 10 architecture. To put these results in perspective, when mapping the VTR and Koios benchmarks to the VTR flagship architecture with a full local LB crossbar, the packing stage constitutes only 29% (3 mins) of the end-to-end VPR runtime (9 mins) on average. This clearly highlights the significant impact of modeling complex LB local interconnect and LE structures on packing runtime, which hinders further exploration of such architectures that resemble modern commercial devices in VPR.

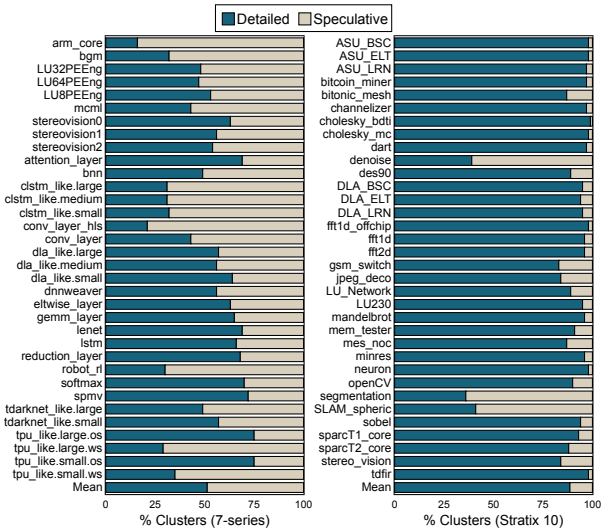

Fig. 3. Breakdown of the percentage of clusters that succeeded with speculative packing vs. clusters that required detailed packing. On average, 51% and 89% of the clusters required detailed packing in the case of VTR and Koios benchmarks on 7-series (left) and Titan32 and Titanium25 benchmarks on Stratix 10 (right), respectively.

To understand the reason for this significant runtime increase, we analyze the packing behavior of all the circuits we experiment with to determine the percentage of logic clusters for which speculative packing (see Section II-B) failed to find a legal solution. Fig. 3 shows that 51% and 89% of all clusters failed the speculative packing step and resorted to detailed packing to find a legal solution for the 7-series and Stratix 10 architectures, respectively. The Stratix 10 architecture generally shows a smaller percentage of speculative packing success due to its more complicated LB compared to the 7-series

[1]BLIFs of the Titan23 and Titanium25 benchmarks are sourced from the Titan flow which uses Quartus as its synthesis frontend, and therefore are only compatible with the VTR Stratix IV and Stratix 10 architecture captures. Conversely, evaluating the VTR and Koios benchmarks on the VTR Stratix 10 capture would require migrating and synthesizing them for Stratix 10 in Quartus to obtain their VTR-compatible BLIFs, which we believe is out of the scope of this work.

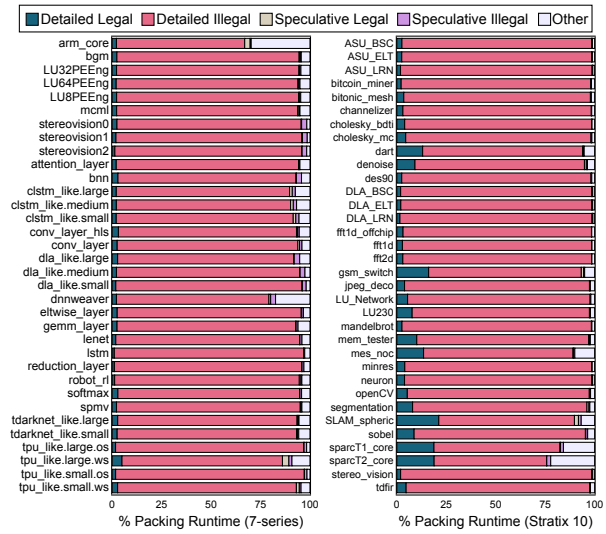

Fig. 4. Breakdown of the the packing runtime for the VTR and Koios benchmarks on the 7-series architecture (left) and Titan23 benchmarks on the Stratix 10 architecture (right). On average, 91% and 90% of the packing runtime is spent on intracluster routing of failed detailed packing attempts.

architecture, emphasizing that speculative packing becomes less effective as LB architecture increases in complexity.

Fig. 4 presents the breakdown of packing runtimes into the time spent on failed and successful attempts of speculative and detailed packing. It shows that, on average, 90% of the packing runtime is attributed to the detailed packing step failing to route clusters, and partial clusters, that have no feasible routing solution. Note that even for benchmarks where clusters that fail speculative packing make up the minority (such as `arm_core` in Fig. 3), detailed packing still overshadows speculative packing in terms of total runtime (as seen in Fig. 4). The reason for this is that while successful speculative packing must only route a cluster once, clusters that fall back to detailed packing will commonly require tens or hundreds of routing calls to reach a solution on the 7-series architecture; this estimate increases by one to two orders of magnitude for Stratix 10. In the absolute worst case, individual clusters in our experiments have seen intracluster router invocation counts up to $2,254$ and $12,133$ on 7-series and Stratix 10 architectures, respectively. These results indicate that significant runtime gains could be realized if the number of intracluster routing attempts made for these unroutable cluster candidates is reduced. In Section IV, we demonstrate that the majority of these problematic cluster patterns are repeats of previously evaluated packing signatures.

IV. Packing Signature Tree

A. Data Structure Overview

To memoize LB packing patterns, we require an approach for comparing graphs of partially or completely packed clusters for equivalency. In its general form, this is a computationally complex graph isomorphism problem; however, our PST data structure leverages properties of greedy packing algorithms to create tree-based packing signature encodings that can be compared in roughly constant time.

The deterministic greedy behavior of common packing algorithms means that repetitions of identical substructures within a technology-mapped netlist (which we postulate are

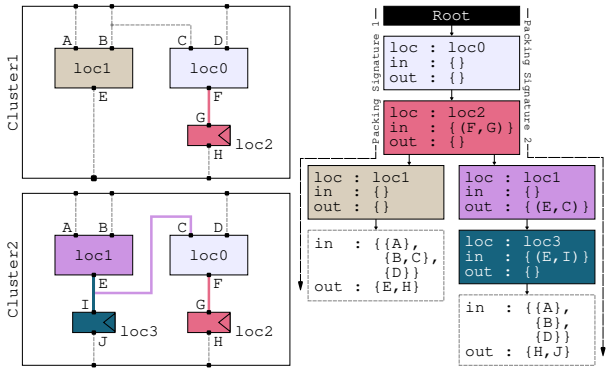

Fig. 5. Example PST representation with two packing signatures (right) corresponding to two packed clusters targeting a simplified LB architecture consisting of two 2-LUTs and two bypassable FFs (left).

abundant in hierarchical digital circuits) will have equivalent affinity metrics computed for their constituent molecules. Consequently, we expect that such identical substructures will receive identical treatment during packing, resulting in equivalent packing patterns. We also assume that a consistent ordering is imposed on the insertion of molecules into equivalent clusters. Under these conditions, which we verify empirically in the next subsection, detecting packing pattern equivalency can be implemented efficiently as traversal over a tree data structure where each node describes a packed primitive and its connectivity. We define each path from the tree root to a leaf node to represent a unique packing signature.

The example in Fig. 5 illustrates our PST data structure in the case of a simple hypothetical LB architecture that consists of two 2-input LUTs and two bypassable FFs. This means there are four possible locations for netlist primitives in a cluster (`loc0`-`loc3`). Although the primitive type that can be inserted in each location is hinted at in the figure (`loc0` and `loc1` for the LUTs, and `loc2` and `loc3` for the FFs), this information is not necessary to uniquely differentiate signatures and therefore is not included in our representation. In addition, each pin for each of these cluster locations is given a unique identifier (`A` through `J` in the figure). The left side of the figure depicts two different cluster candidates formed by the packing algorithm, while the right side shows how the packing signatures of these two clusters are represented in our PST. We note that VPR represents cluster resources that support M modes of operation as M distinct mutually exclusive locations (e.g., a LUT-6 that can be fractured into two LUT-5 is represented as 3 locations, each with distinct pins). Therefore, no special consideration is needed for operating modes in our representation.

Assume that `Cluster1` was the first cluster to be packed by first inserting a LUT in `loc0`, then a FF in `loc2`, and finally a LUT in `loc1`. This sequence of packing decisions adds 3 *location and connectivity nodes (LCNs)* to the PST, followed by an *external connectivity node (ECN)* added to the *tail* of the path labeled `Packing Signature 1` in Fig. 5. When a molecule's atoms are packed into a cluster, their corresponding LCNs store the placement location as well as the internal cluster connections that drive their input pins and are driven by their output pins.

Since the packing algorithm sequentially adds molecules to a cluster, it is impossible to know at the time of packing a molecule if its connections entering/exiting the cluster will

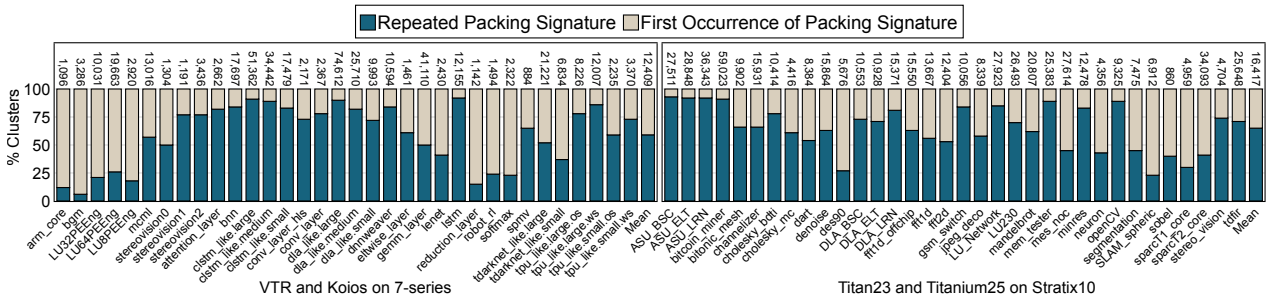

Fig. 6. Breakdown of final packed clusters that are the first occurrence vs. have a repeated packing signature. On average, 59% and 65% of the clusters have repeated signatures for VTR and Koios benchmarks on 7-series (left) and Titan23 and Titanium25 benchmarks on Stratix 10 (right), respectively. The total number of clusters for each benchmark is shown above its entry.

remain as external connections (i.e., that the other molecules driving or driven by it will not be packed into the same cluster later). To avoid backtracking through the PST, only pin connections that are known to be fully contained within the cluster upon the addition of an atom are listed in its corresponding LCN; if the newly packed atom is driven by a pin belonging to a previously packed atom (e.g., `loc2` FF driven by `loc0` LUT in Fig. 5), then a tuple with the internal source and sink pin identifiers (e.g., `(F,G)`) is added to the set of input connections in the FF's LCN.

Because the connections entering and leaving the cluster are not fully known until the cluster is finalized, their description is included in an ECN at the end of a packing signature. An ECN stores the internal input pins driven by external cluster inputs as a set of sets, where the union of the inner sets includes all pins that are driven by a source outside of the cluster, and pins belonging to the same inner set share the same driver. To illustrate this, pins `B` and `C` of `Cluster1` in Fig. 5 share the same external source, and pins `A` and `D` have independent sources. Therefore, the ECN at the end of the `Packing Pattern 1` branch of the PST has `B` and `C` in the same inner set. The ECN also stores a set of all of the internal pins that drive nets exiting the cluster.

If `Cluster2` is then packed in the following sequence: LUT in `loc0`, FF in `loc2`, LUT in `loc1`, and FF in `loc3`, the PST is traversed starting from the root as each new molecule is added to the cluster. Since the first two steps are identical to those done during packing `Cluster1`, no new LCNs are added to the tree. However, the third and fourth steps diverge from `Packing Pattern 1` and add two new LCNs followed by an ECN, as shown by the branch labeled `Packing Pattern 2` in Fig. 5. When packing a new cluster, if the PST is traversed from its root to an ECN leaf without adding any new nodes, this means that an identical cluster has been packed before. *Déjà vu!*

Modern FPGA CAD tools, including VPR, will recognize that input pins to LEs and certain other primitive types are logically equivalent and, therefore, interchangeable. Logically, it is sufficient for packing signatures to represent equivalent inputs as a single, shared pin identifier; however, there are some implementation caveats with this approach. Heuristics based routing algorithms used by FPGA CAD tools for local and global routing, such as PathFinder [29], can occasionally fail to route a collection of signals when a valid routing exists [30]. Our experiments found that in rare instances, logically equivalent packing signatures could report being simultaneously legal and illegal if permutations in equivalent primitive inputs seeded the router to produce different results, and at least one routing failed. To address this issue, we choose not to map equivalent pins to the same pin identifier in our PST implementation in VPR. Instead, we simply generate LCNs using the non-random placeholder pin mappings that the technology mapper assigns to primitives with equivalent pins when constructing the netlist.

B. Packing Signature Characterization

We first implement our PST data structure in VPR and use it to enumerate the occurrences of packing signatures for the FPGA architectures and benchmarks evaluated in Section III-B. For this experiment, we modify the PST by adding a counter variable to each ECN which gets incremented every time that ECN is visited (i.e., a legal cluster is finalized). Fig. 6 summarizes the results of this experiment with the total number of logic clusters for each benchmark reported on top of the bars. It shows the ratio of unique signatures out of the final packed clusters of each circuit (i.e., count of signature first occurrences) in comparison to repeated signatures. On average, 59% and 65% of the clusters packed for the evaluated benchmarks have repeated packing signatures on the 7-series and Stratix 10 architectures, respectively. Of the tested benchmarks, approximately one-fifth (13 circuits from the Koios, Titan23, and Titanium25 suites) exhibited more than 85% repeated clusters. Nine of these benchmarks come from the deep learning (DL) domain, which is known to have highly regular designs that feature large arrays of repeated compute structures. In contrast, 53% of clusters from non-DL Titan23, Titanium, and VTR benchmarks are repeated, on average.

V. Déjà vu Packing

The results presented in the previous section establish three key observations. On average across a diverse set of benchmarks targeting two VTR architectures based on modern commercial FPGAs: (1) the packing stage constitutes 58–94% of VPR's end-to-end runtime, depending on the architecture, (2) upwards of 90% of this packing runtime is spent performing intracluster routing legality checks on illegal clusters, and partial clusters, during detailed packing, and (3) 59–65% of the final packed clusters have packing signatures that have been previously encountered during the packing process. The runtime impact from this last point is compounded by the numerous partially packed cluster patterns that must be evaluated for routability during intermediate steps of detailed packing (see Section II-B), many of which may have been seen before.

To eliminate redundant legality checks, we update the packer to generate and insert an ECN into the PST each time

TABLE I. Déjà Vu runtime gains for the packing stage and end-to-end VPR flow.

VTR & Koios Benchmarks on 7-series				
	Packing Runtime (mins)		VPR Runtime (mins)	
Benchmark	**Baseline**	**Déjà Vu**	**Baseline**	**Déjà Vu**
arm_core	1.2	1.0 (1.3×)	2.3	2.0 (1.1×)
bgm	10.7	7.2 (1.5×)	15.1	11.9 (1.3×)
LU32PEEng	41.2	21.4 (1.9×)	59.6	44.0 (1.4×)
LU64PEEng	69.1	36.0 (1.9×)	117.4	87.2 (1.3×)
LU8PEEng	13.1	7.5 (1.7×)	16.8	11.8 (1.4×)
mcml	28.9	17.8 (1.6×)	65.4	59.4 (1.1×)
stereovision0	6.0	1.7 (3.5×)	7.1	2.7 (2.6×)
stereovision1	5.5	0.4 (13.0×)	7.8	2.6 (3.0×)
stereovision2	16.5	2.0 (8.3×)	22.5	9.8 (2.3×)
attention_layer	14.6	1.8 (8.0×)	39.3	30.7 (1.3×)
bnn	24.7	5.3 (4.7×)	59.9	48.4 (1.2×)
clstm_like.large	81.3	13.8 (5.9×)	211.8	154.7 (1.4×)
clstm_like.medium	55.9	9.7 (5.8×)	131.4	93.9 (1.4×)
clstm_like.small	31.3	6.0 (5.2×)	60.3	42.0 (1.4×)
conv_layer_hls	2.4	1.3 (1.9×)	8.4	7.4 (1.1×)
conv_layer	4.5	0.8 (5.6×)	7.9	4.1 (1.9×)
dla_like.large	169.5	29.6 (5.7×)	462.0	345.5 (1.3×)
dla_like.medium	86.2	15.7 (5.5×)	152.6	95.5 (1.6×)
dla_like.small	52.7	10.3 (5.1×)	70.3	35.5 (2.0×)
dnnweaver	20.8	7.3 (2.8×)	51.0	42.7 (1.2×)
eltwise_layer	7.2	2.2 (3.3×)	9.51	4.6 (2.1×)
gemm_layer	103.1	31.5 (3.3×)	191.5	130.8 (1.5×)
lenet	8.1	3.6 (2.2×)	10.9	6.7 (1.6×)
lstm	80.1	6.8 (11.9×)	106.0	40.0 (2.7×)
reduction_layer	8.5	4.6 (1.8×)	9.8	6.0 (1.6×)
robot_rl	3.9	0.8 (4.6×)	5.6	2.6 (2.2×)
softmax	9.8	5.0 (1.9×)	12.1	7.7 (1.6×)
spmv	3.9	0.9 (4.2×)	9.6	6.8 (1.4×)
tdarknet_like.large	66.9	24.4 (2.7×)	133.0	95.2 (1.4×)
tdarknet_like.small	27.1	12.5 (2.2×)	75.4	67.4 (1.1×)
tpu_like.large.os	69.6	5.2 (13.4×)	96.0	39.1 (2.5×)
tpu_like.large.ws	9.6	3.0 (3.2×)	40.8	38.2 (1.1×)
tpu_like.small.os	20.6	2.5 (8.1×)	26.9	10.6 (2.5×)
tpu_like.small.ws	5.8	1.8 (3.2×)	13.7	10.3 (1.3×)
Geomean		**(3.75×)**		**(1.57×)**

Titan23 & Titanium25 Benchmarks on Stratix 10				
	Packing Runtime (mins)		VPR Runtime (mins)	
Benchmark	**Baseline**	**Déjà Vu**	**Baseline**	**Déjà Vu**
ASU_BSC	2316.2	79.1 (29.3×)	2407.1	196.9 (12.2×)
ASU_ELT	2460.1	87.9 (28.0×)	2727.9	405.1 (6.7×)
ASU_LRN	3782.1	197.9 (19.1×)	3839.9	262.7 (14.6×)
bitcoin_mine	4463.6	198.0 (22.5×)	4610.4	349.6 (13.2×)
bitonic_mesh	461.9	27.0 (17.1×)	476.2	46.7 (10.2×)
channelizer	1288.4	126.1 (10.2×)	1319.6	166.3 (7.9×)
cholesky_bdti	747.0	73.0 (10.2×)	755.6	84.6 (8.9×)
cholesky_mc	325.2	61.3 (5.3×)	329.7	66.1 (5.0×)
dart	154.6	43.0 (3.6×)	163.4	53.3 (3.1×)
denoise	169.4	60.7 (2.8×)	215.6	110.1 (2.0×)
des90	392.5	67.0 (5.9×)	399.7	74.9 (5.3×)
DLA_BSC	1122.1	129.1 (8.7×)	1131.3	141.4 (8.0×)
DLA_ELT	1124.1	140.6 (8.0×)	1133.5	152.8 (7.4×)
DLA_LRN	1858.6	140.5 (13.2×)	1872.0	159.9 (11.7×)
fft1d_offchip	1213.2	141.5 (8.6×)	1243.1	178.7 (7.0×)
fft1d	1182.6	122.4 (9.7×)	1199.2	144.9 (8.3×)
fft2d	1010.7	113.7 (8.9×)	1028.6	137.8 (7.5×)
gsm_switch	134.7	24.8 (5.4×)	150.9	43.4 (3.5×)
jpeg_deco	424.5	105.4 (4.0×)	443.6	125.7 (3.5×)
LU_Network	1046.9	89.3 (11.7×)	1085.1	138.3 (7.8×)
LU230	845.0	131.3 (6.4×)	882.2	179.3 (4.9×)
mandelbrot	1828.4	191.8 (9.5×)	1851.3	220.6 (8.4×)
mem_tester	527.2	42.0 (12.6×)	550.8	72.2 (7.6×)
mes_noc	421.4	118.8 (3.5×)	466.5	169.1 (2.8×)
minres	680.3	58.3 (11.7×)	689.6	70.6 (9.8×)
neuron	380.1	97.3 (3.9×)	383.1	100.4 (3.8×)
openCV	383.0	29.5 (13.0×)	395.1	43.6 (9.1×)
segmentation	79.1	42.5 (1.9×)	93.4	58.1 (1.6×)
SLAM_spheric	29.0	23.0 (1.3×)	42.0	37.5 (1.1×)
sobel	29.2	15.0 (1.9×)	32.6	18.4 (1.8×)
sparcT1_core	55.5	43.0 (1.3×)	62.3	50.3 (1.2×)
sparcT2_core	355.6	211.3 (1.7×)	407.5	262.0 (1.6×)
stereo_vision	474.6	69.5 (6.8×)	481.9	78.2 (6.2×)
tdfir	1278.7	130.6 (9.8×)	1310.7	171.2 (7.7×)
Geomean		**(6.92×)**		**(5.31×)**

the router is invoked to assess cluster routing legality. The legality check outcome then gets stored in the ECN as an additional boolean field. Now, when the packer requires the legality status of a cluster it is packing, it will first search the child ECNs of the LCN at the tail of the active packing signature. If an ECN matches the current external connectivity state of the cluster, then the legality status recorded in that ECN is provided to the packer; otherwise, the legality status is unknown and the router must be invoked.

When an illegal cluster pattern is identified during detailed packing, the packer will remove the most recent molecule (which caused the violation) from the cluster. In turn, the pointer to the tail of the active packing signature must also be reset to its location prior to the addition of the most recent molecule. However, the LCN(s) and ECN that were added to the PST for the violating molecule remain. This way, subsequent detailed packing attempts will find the ECN, and, seeing that is is marked as illegal, know to skip invoking the router if this pattern is reached again.

The modified packing procedure can thus be summarized as follows. First, speculative packing is attempted as before. As molecules are added to the cluster, the PST is traversed by following the corresponding LCNs, if they exist. Otherwise, new LCNs are added, creating new branches in the tree. Because speculative packing invokes routing just once when a fully packed cluster candidate has been formed, only one ECN gets generated and appended to the end of speculative packing signatures. In the event that speculative packing fails and detailed packing is performed, an ECN is created for every molecule packed in the detailed pass and is used to store the legality result of the routing call made for the molecule. In any state where a child ECN of the active packing signature's tail matches the external connectivity of the cluster candidate being packed, the packer can read the legality check outcome from this ECN instead of incurring the cost of performing intracluster routing.

VI. Experimental Results

Table I shows the runtime gains achieved by Déjà Vu packing for the VTR and Koios benchmarks on the 7-series architecture, as well as for the Titan23 and Titanium25 benchmarks on Stratix 10. Our Déjà Vu packing technique results in significant runtime improvements across every benchmark, while preserving identical QoR to the baseline VPR since the packing solution is identical.

The 7-series benchmarks see speedups of up to 13.4× in the packing stage and 3.0× for the entire VPR flow. For the two Koios benchmarks with the longest runtimes, `dla_like.large` and `clstm_like.large`, the packing stage is 5.7× and 5.9× faster, resulting in VPR runtime reductions of 1.3× and 1.4×, respectively. To put this in perspective, the total runtime of of `dla_like.large` is reduced by 1.9 hours, bringing the longest runtime across all

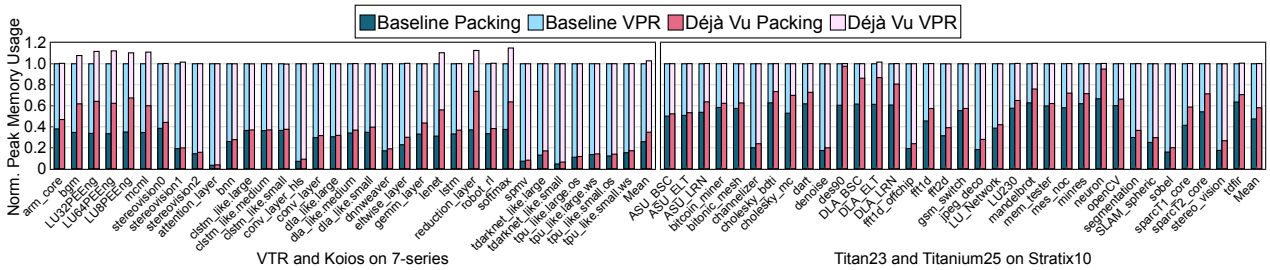

Fig. 7. Comparison of peak memory utilization of the VPR packing stage and full VPR flow with and without déjà vu packing. All values are normalized to the baseline peak VPR memory utilization. Packing memory utilization increased by 30% and 24% on average for the VTR and Koios benchmarks on 7-series (left) and Titan23 and Titanium25 benchmarks on Stratix 10 (right). VPR peak memory utilization increased by 3% and <1% on average.

7-series benchmarks from 7.7 hours down to 5.8 hours. On average, total VPR runtime is reduced by 1.6×.

Our baseline runtime measurements for the Stratix 10 capture show that modeling modern commercial LB architectures in VTR results in a significant overhead in packing runtime. On average across the Titan23 and Titanium25 benchmarks, packing takes 16.2 hours, with the largest circuit, `bitcoin_miner`, spending 74.4 hours out of the 76.8-hour end-to-end runtime in the packing stage. As a result, a faithful Stratix 10 capture becomes impractical to use for architecture and CAD algorithm exploration. The introduction of Déjà Vu packing yields runtime reductions of up to 29.3× in the packing stage. This brings the maximum VPR runtime across all Stratix 10 benchmarks down from 76.8 hours to around 6.8 hours, and the average total VPR runtime to 2.2 hours down from 16.8 hours, well within acceptable limits for mapping large circuits in modern FPGA CAD tools. On average, for the Stratix 10 architecture, **Déjà Vu packing speeds up the entire VPR flow by 5.3×**.

Managing the PST structure in the packing stage introduces very little overhead. We measured that insertion and lookup into the PST accounts for only 0.8% of total packing runtime on average (with a maximum of 1.6%) across all of the evaluated 7-series and Stratix 10 benchmarks. Fig. 7 compares the peak memory usage for the packing stage and the full VPR flow with and without our PST data structure. On average, the inclusion of a PST for Déjà Vu packing results in a 27% increase in peak memory utilization for the packer. From the measured memory deltas between the baseline and Déjà Vu packing stages, we estimate that PST sizes range from 19–1258 MB for the 7-series benchmarks, and 21–657 MB for the Stratix 10 benchmarks. The PST structure is freed at the end of the packing stage, prior to later VPR stages which use greater peak memory, thus the peak VPR memory usage is generally unaffected by Déjà Vu packing. Having verified the absence of memory leaks with Valgrind [31], the peak VPR memory increases that can be seen in a number of the 7-series benchmarks are best attributed to underlying memory allocator behavior, and not persisting PST structures.

Déjà Vu packing is most useful for architectures with complex LB structures that result in a significant amount of legality checks during detailed packing. However, we also evaluated its use on the VTR flagship architecture that has a simple LB with a full local crossbar. In such architectures, the overwhelming majority of their packed clusters succeed with speculative packing, since any candidate packed cluster is legal if it uses no more than the available LB pins. We confirm that Déjà Vu packing does not introduce any runtime overhead for these architectures; the packing stage is 3% faster on average across the VTR and Koios benchmarks on the flagship architecture, which does not result in any noticeable change in the end-to-end runtime. In addition, since the memory footprint of the PST scales primarily with failed legalization checks (due to the ECNs generated by detailed packing), an average of only 0.8% increase in peak packing memory utilization (with a maximum of 2.6%) is measured for the VTR flagship architecture. This is demonstrates the desirable property that the memory overhead of Déjà Vu packing is tightly coupled to the runtime savings that it can achieve.

VII. Conclusion

We presented Déjà Vu packing, a runtime optimization for FPGA packing algorithms, particularly for architectures with complex logic block (LB) internals that resemble contemporary commercial devices. Through detailed runtime analysis of the VPR tool, we demonstrated that packing dominates the overall flow runtime for such architectures, and that this runtime is predominantly spent performing legality checks that evaluate the routing feasibility of candidate clusters. Our key insight is that, on average, 62% of the final packed clusters in the benchmarks we evaluated are recurring packing patterns, each of which can invoke numerous costly legality checks on intermediate clustering steps. We introduced a novel packing signature tree (PST) data structure, which enables efficient memoization and comparison of cluster packing patterns. By integrating the PST into VPR's packing algorithm, we enable the packer to recognize packing patterns that have been seen before (i.e., déjà vu instances) and skip their redundant legality checks by reusing previously computed outcomes. We evaluated our approach across 68 benchmarks from the VTR, Koios, Titan23, and Titanium suites, targeting AMD 7-series and Altera Stratix 10 VTR architecture captures. Our results demonstrate substantial runtime improvements: up to 29.3× reduction in packing time, with average speedups of 5.1× for packing and 2.9× for the end-to-end VPR flow. These runtime gains are achieved while fully preserving the quality of results, as our approach does not alter the optimization decisions of the packing algorithm. This work is being integrated into the VPR open-source tool to accelerate architecture and CAD algorithm exploration for modern FPGAs with complex LB structures.

Acknowledgment

This work was funded by the Natural Sciences and Engineering Research Council of Canada (NSERC) and the University of Waterloo President's Research Award. The authors would like to thank Alexandre Singer, Amirhossein Poolad, and Vaughn Betz for their insightful feedback during integrating this work into the VTR master branch.

REFERENCES

[1] W. Li and D. Z. Pan, "A New Paradigm for FPGA Placement Without Explicit Packing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 38, no. 11, pp. 2113–2126, 2019.

[2] K. E. Murray, J. Luu, M. J. Walker, C. McCullough, S. Wang, S. Huda, B. Yan, C. Chiasson, K. B. Kent, J. Anderson *et al.*, "Optimizing FPGA Logic Block Architectures for Arithmetic," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 6, pp. 1378–1391, 2020.

[3] M. A. Elgammal, A. Mohaghegh, S. G. Shahrouz, F. Mahmoudi, F. Koşar, K. Talaei, J. Fife, D. Khadivi, K. Murray, A. Boutros *et al.*, "VTR 9: Open-source CAD for Fabric and Beyond FPGA Architecture Exploration," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 18, no. 3, pp. 1–53, 2025.

[4] A. Arora, A. Boutros, S. A. Damghani, K. Mathur, V. Mohanty, T. Anand, M. A. Elgammal, K. B. Kent, V. Betz, and L. K. John, "Koios 2.0: Open-Source Deep Learning Benchmarks for FPGA Architecture and CAD Research," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 42, no. 11, pp. 3895–3909, 2023.

[5] K. E. Murray, S. Whitty, S. Liu, J. Luu, and V. Betz, "Titan: Enabling Large and Complex Benchmarks in Academic CAD," in *IEEE International Conference on Field programmable Logic and Applications (FPL)*, 2013, pp. 1–8.

[6] V. Betz and J. Rose, "Cluster-based Logic Blocks for FPGAs: Area-Efficiency vs. Input Sharing and Size," in *IEEE Custom Integrated Circuits Conference (CICC)*, 1997, pp. 551–554.

[7] A. Marquardt, V. Betz, and J. Rose, "Using Cluster-Based Logic Blocks and Timing-Driven Packing to Improve FPGA Speed and Density," in *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, 1999, pp. 37–46.

[8] E. Bozorgzadeh, S. Ogrenci-Memik, and M. Sarrafzadeh, "RPack: Routability-Driven Packing for Cluster-Based FPGAs," in *ACM/IEEE Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2001, pp. 629–634.

[9] A. Singh, G. Parthasarathy, and M. Marek-Sadowska, "Efficient Circuit Clustering for Area and Power Reduction in FPGAs," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 7, no. 4, pp. 643–663, 2002.

[10] D. T. Chen, K. Vorwerk, and A. Kennings, "Improving Timing-Driven FPGA Packing with Physical Information," in *IEEE International Conference on Field Programmable Logic and Applications (FPL)*, 2007, pp. 117–123.

[11] S. T. Rajavel and A. Akoglu, "MO-Pack: Many-Objective Clustering for FPGA CAD," in *ACM/IEEE Design Automation Conference (DAC)*, 2011, pp. 818–823.

[12] M. Tom, D. Leong, and G. Lemieux, "Un/DoPack: Re-clustering of Large System-on-Chip Designs with Interconnect Variation for Low-Cost FPGAs," in *IEEE/ACM International Conference on Computer-aided design (ICCAD)*, 2006, pp. 680–687.

[13] H. Liu and A. Akoglu, "T-NDPack: Timing-Driven Non-Uniform Depopulation Based Clustering," in *IEEE Southern Conference on Programmable Logic (SPL)*, 2009, pp. 9–14.

[14] G. Karypis and V. Kumar, "Multilevel K-Way Hypergraph Partitioning," in *ACM/IEEE Design Automation Conference (DAC)*, 1999, pp. 343–348.

[15] W. Feng, "K-Way Partitioning Based Packing for FPGA Logic Blocks Without Input Bandwidth Constraint," in *IEEE International Conference on Field-Programmable Technology (FPT)*, 2012, pp. 8–15.

[16] W. Feng, J. Greene, K. Vorwerk, V. Pevzner, and A. Kundu, "Rent's Rule Based FPGA Packing for Routability Optimization," in *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2014, pp. 31–34.

[17] D. Vercruyce, E. Vansteenkiste, and D. Stroobandt, "Runtime-Quality Tradeoff in Partitioning Based Multithreaded Packing," in *IEEE International Conference on Field Programmable Logic and Applications (FPL)*, 2016, pp. 1–9.

[18] ——, "How Preserving Circuit Design Hierarchy During FPGA Packing Leads to Better Performance," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 37, no. 3, pp. 629–642, 2017.

[19] L. Singhal, M. A. Iyer, and S. Adya, "LSC: A Large-Scale Consensus-Based Clustering Algorithm for High-Performance FPGAs," in *ACM/IEEE Design Automation Conference (DAC)*, 2017, pp. 1–6.

[20] G. Chen and J. Cong, "Simultaneous Timing Driven Clustering and Placement for FPGAs," in *International Conference on Field Programmable Logic and Applications (FPL)*, 2004, pp. 158–167.

[21] M. A. Elgammal and V. Betz, "Breaking Boundaries: Optimizing FPGA CAD with Flexible and Multi-threaded Re-Clustering," in *ACM International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies (HEART)*, 2023, pp. 11–18.

[22] J. Luu, J. Anderson, and J. Rose, "Architecture description and packing for logic blocks with hierarchy, modes and complex interconnect," *Proceedings of 2011 Acm/Sigda International Symposium on Field Programmable Gate Arrays*, pp. 227–236, 2011.

[23] J. Luu, J. Rose, and J. Anderson, "Towards Interconnect-Adaptive Packing for FPGAs," in *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2014, pp. 21–30.

[24] G. Lemieux and D. Lewis, "Using Sparse Crossbars within LUT," in *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2001, pp. 59–68.

[25] T. D. Haroldsen, "Academic Packing for Commercial FPGA Architectures," Ph.D. dissertation, Brigham Young University, 2017.

[26] 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*, vol. 7, no. 3, pp. 1–30, 2014.

[27] K. E. Murray, O. Petelin, S. Zhong, J. M. Wang, M. Eldafrawy, J.-P. Legault, E. Sha, A. G. Graham, J. Wu, M. J. P. Walker, H. Zeng, P. Patros, J. Luu, K. B. Kent, and V. Betz, "VTR 8: High-performance CAD and Customizable FPGA Architecture Modelling," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 13, no. 2, pp. 1–60, 2020.

[28] K. T. Khoozani, A. A. Dehkordi, and V. Betz, "Titan 2.0: Enabling Open-Source CAD Evaluation with a Modern Architecture Capture," in *IEEE International Conference on Field-Programmable Logic and Applications (FPL)*, 2023, pp. 57–64.

[29] L. McMurchie and C. Ebeling, "PathFinder: A Negotiation-Based Performance-Driven Router for FPGAs," in *ACM International Symposium on Field-Programmable Gate Arrays (FPGA)*, 1995, pp. 111—-117.

[30] S. Shrivastava, S. Nikolić, S. Tanaka, C. Ravishankar, D. Gaitonde, and M. Stojilović, "Guaranteed Yet Hard to Find: Uncovering FPGA Routing Convergence Paradox," in *IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2025, pp. 143–151.

[31] "Valgrind," https://valgrind.org, accessed: 2026-01-17.