2024 34th International Conference on Field-Programmable Logic and Applications (FPL) | 979-8-3315-3007-5/24/$31.00 ©2024 IEEE | DOI: 10.1109/FPL64840.2024.00016

Better Together: Combining Analytical and Annealing Methods for FPGA Placement

Rachel Selina Rajarathnam[†], Kate Thurmer[*], Vaughn Betz[*], Mahesh A. Iyer[‡] and David Z. Pan[†]
[†]Dept. of Electrical and Computer Engineering, The University of Texas at Austin, Austin, TX, USA
[*]Dept. of Electrical and Computer Engineering, University of Toronto, Toronto, ON, Canada
[‡]Intel Corporation, San Jose, CA, USA
rachelselina.r@utexas.edu[†], {kate.thurmer@mail, vaughn@ece}.utoronto.ca[*], mahesh.iyer@intel.com[‡], dpan@ece.utexas.edu[†]

***Abstract*—Placement is a critical step in the FPGA design implementation flow that strongly impacts routability and timing closure. Recent state-of-the-art academic analytical placers have achieved impressive scalability but are limited to AMD Ultrascale-like architectures and mostly synthetic designs. On the other hand, *VPR*, the place and route tool within the widely used open-source Verilog-to-Routing (*VTR*) toolchain, can produce a legal placement for any arbitrary architecture; however, its simulated annealing placer scales poorly. Thus, there is a clear need to bring scalable, high-quality placement to realistic architectures and circuits. In this work, we develop a hybrid framework that combines the strength of a scalable flat analytical placer with the flexibility of simulated annealing techniques to adapt to various architectures and circuits, substantially improving the quality of results. We augment the state-of-the-art analytical *elfPlace* FPGA placer as *aug-elfPlace*, generalizing its architecture modeling to handle real-world constraints and target different and more complete architectures. We leverage *VPR*'s legalization capability to integrate with external placers such as *aug-elfPlace*. *VPR*'s simulated annealing placer can further optimize the legalized placement, and *VPR*'s router and timing analysis can provide final quality results. By integrating wirelength-driven *aug-elfPlace* and *VPR*, our hybrid framework achieves up to 2% timing improvement with 15% reduction in routed wirelength compared to timing-driven *VPR*, on average across the large and heterogeneous Titan23 benchmark suite targeting an Intel Stratix-IV-like architecture.**

***Index Terms*—FPGA Placement, VTR, CAD**

I. Introduction

Modern high-performance FPGAs employed in data centers and the cloud consist of heterogeneous resources such as digital signal processors (DSPs), IO (input/output), and memory blocks, in addition to re-configurable logic elements such as look-up tables (LUTs) and flip-flops (FFs) [1]. During the FPGA design implementation flow, the placement stage – during which the physical locations of the heterogeneous design instances on the FPGA layout are determined, plays a decisive role in design routability and timing closure.

Through public benchmarking, FPGA placement contests continue to advance the research of scalable placement algorithms [2]–[4]. Recent academic analytical placers have attained remarkable scalability on mostly synthetic contest designs through innovations in placement formulations, accelerating portions of the placer using multi-thread CPUs, GPUs, and FPGAs [5]–[14], and incorporating timing optimizations [15]–[19]. However, most academic advancements on scalable FPGA placement remain limited to AMD Ultrascale and related architectures [20].

On the other hand, the widely used Verilog-to-Routing (*VTR*) is a flexible open-source FPGA toolchain that can target complex, realistic devices [21] and has been used to explore and develop novel architectures [22]. *VPR*, the place and route tool within *VTR*, can produce a legal placement for a wide range of input FPGA architectures with timing optimization. However, *VPR*'s simulated annealing (SA) based placer scales poorly and thus must rely on clustering to limit the size and complexity of the placement problem.

In this work, we develop an integrated hybrid framework to enable scalable and flexible FPGA placement by leveraging the strengths of both flat analytical placement and SA refinement. Among the flat analytical academic placers, *elfPlace* [8] employs a non-linear electrostatics-based placement formulation, and several state-of-the-art placement frameworks have been built on *elfPlace* algorithms [9], [11]. As the first step toward an integrated framework, we introduce *aug-elfPlace*, a generalization of *elfPlace* that can accommodate complex architectural constraints (such as carry chains) and can thus target FPGA architectures beyond the AMD Ultrascale architecture [20].

To enable interoperability between *aug-elfPlace* and *VTR*, we use *VPR* to pre-process the architecture and design information to allow benchmarking for *aug-elfPlace*. Leveraging *VPR*'s legalizer [23], the flat placement solution from *aug-elfPlace* is verified and reconstructed to fix any legality errors, thereby allowing for placement refinement, routing, and validation in *VPR*. By further refining *aug-elfPlace*'s placement solutions using *VPR*'s simulated annealer, the proposed hybrid framework significantly improves the quality of results.

The integrated framework contributes to the open-source FPGA physical design ecosystem [24] to enable the community to build on it to develop high-performance placement and routing algorithms that apply to a wide range of FPGA architectures. While the proposed integrated framework can be enhanced to target any FPGA architecture, we target an Intel Stratix-IV-like architecture on the large Titan23 benchmark suite [25]. The primary contributions of this work are:

- An open-source hybrid placement framework that combines the scalability of a flat analytical placer with the flexibility of *VTR* to significantly improve the quality of results (QoR).

- An interface, consisting of a *VPR* legalizer [23], that enables external placement tools to integrate with *VPR* for placement refinement, routing, and validation.
- We generalize *elfPlace*'s architecture modeling and augment its algorithms in several ways, including developing a scalable auction algorithm to legalize large DSP/memory instances and designing new heuristics to optimize memory LABs (MLABs) and efficiently handle carry chains.
- With a flat analytical placer (*aug-elfPlace*) and *VPR*'s simulated annealer, the proposed hybrid framework achieves up to 2% lower critical path delay (CPD) with 15% routed wire length reduction on average, cf. *VPR*'s timing-driven placer, on the Titan23 benchmark suite.

Section II provides the required background for this work. Section III details the enhancements in *aug-elfPlace* placer and introduces the proposed integrated framework with *VTR*. Experiments on the Titan23 benchmarks are presented in Section IV, followed by a summary in Section V.

II. Background and Prior Work

A. Prior Block-level Hybrid Placer

HCAS [26], a block-level hybrid analytical placer, employs a quadratic global placement formulation to place packed clusters from the *VPR*'s packer, followed by simulated annealing for legalizing DSP/memory blocks and detailed placement. While HCAS improves upon *VPR*7 in terms of QoR on small benchmarks, the proposed integrated framework, combining flat analytical and simulated annealing-based placers, substantially improves the QoR over *VPR*8+ on the extensive Titan23 benchmarks.

B. The VTR Framework

The Verilog-to-Routing framework (*VTR*) is an open-source, adaptive end-to-end CAD framework for FPGA design implementation and architecture exploration [21]. Given an FPGA architecture description and a design in Verilog HDL, *VTR* first synthesizes the design into *primitives* available on the device (such as LUTs, FFs, and small multipliers) by interfacing with open-source synthesis tools (Yosys and ODIN II) to target a variety of architectures, as well as commercial tools, such as Intel's Quartus. *VTR* is used in a wide variety of architecture investigations (e.g., [27]) and architecture-aware CAD research such as hard network-on-chip (NoC) usage optimization for FPGAs that contain them [28]. *VTR* can also be augmented with a bitstream generator to program commercial or novel FPGAs [29] or pair it with a fabric generator (e.g. [24]) to automatically create an FPGA layout from a *VTR* architecture description.

VTR's place and route tool, *VPR* takes in the technology-mapped netlist of primitives and the FPGA architecture description, packs design primitives into legal soft logic, RAM, DSP and other blocks (*clusters*), places these clusters on *sites* of the appropriate type within the device, and routes them. The placer in *VPR* employs a reinforcement-learning guided simulated annealing-based algorithm and operates in two phases: $pack$ [30] and $place$ [31]. The locations of all the instances during the $pack$ and $place$ stages are legal, thereby avoiding the need for the legalization stage that is generally required in analytical placers.

During the $pack$ stage, the technology-mapped design instances are grouped into legal clusters (that can map to some sites) based on the netlist connectivity and adhering to the FPGA architectural constraints. The $pack$ stage is also crucial to reduce the number of elements processed by the annealer during the $place$ stage, as the annealer does not scale well to large problem sizes. The *VPR* placer can optimize for total estimated wire length or a combination of wire length and critical path delay.

C. Electrostatics-based Analytical Placement

elfPlace is a flat wire length-driven analytical FPGA placer consisting of three phases: a flat global placement (GP), a simultaneous clustering and legalization (LG), and a detailed placement (DP) refinement [8], Using Lagrangian relaxation, the placement formulation is represented as,

$$\min_{x,y} f(x,y) = \min\left(\sum_{e\in E} W_e(x,y) + \lambda.D(x,y)\right), \quad (1)$$

where $W_e(x,y)$ represents the wirelength of net $e \in E$, which depends on the (x,y) locations of all the connected instances. The density term $D(x,y)$ evaluates the overlap among the instances, and λ is a Lagrangian multiplier.

elfPlace employs an electrostatic-system analogy to formulate the density term $D(x,y)$ as the electrostatic system's potential energy $\Phi(x,y)$. Design instances are assigned positive charges proportionate to their areas, and the respective sites have a fixed negative charge. An augmented lagrangian multiplier formulation is employed in *elfPlace*,

$$f(x,y) = W_e(x,y) + \sum_{s\in S} \lambda_s\big(\Phi_s(x,y) + c_s\Phi_s(x,y)^2\big). \quad (2)$$

λ_s and Φ_s represent the density multiplier and electric potential energy for site type s, and c_s is a constant inversely proportionate to initial potential energy. Separate electrostatic systems handle different resource types $s \in \{LUT, FF, DSP, memory\}$.

The DSP and memory instances are legalized at the end of the global placement. Then, the LUTs and FFs are clustered and legalized using a flat consensus-based simultaneous packing-legalization algorithm [32], [33]. Finally, *elfPlace* refines the legal placement using an independent set matching (ISM). Nevertheless, *elfPlace* is tuned to one set of synthetic benchmarks [2] and is not flexible to target a substantially different FPGA architecture.

D. FPGA Architecture

A *Slice* site contains multiple LUTs, FFs, and adders grouped into *subSlices*.

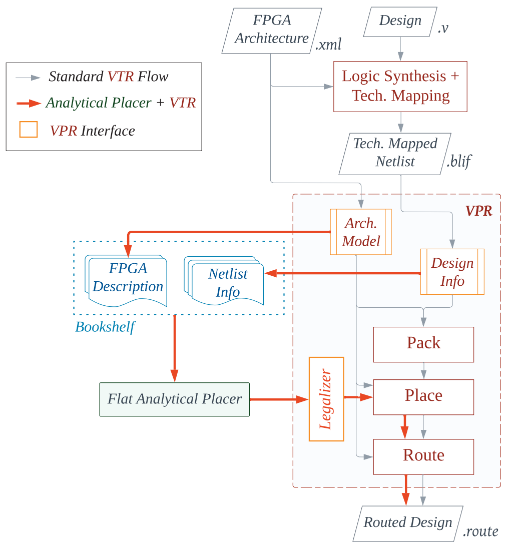


Fig. 1. The proposed hybrid placement flow.

1) Ultrascale: A column-based AMD Ultrascale architecture consists of heterogeneous resource types such as IO (input/output), *Slice*, DSP, and memory sites [20]. A *Slice* or a configurable logic block (CLB) consists of an 8-bit adder and eight *subSlices* or basic logic elements (BLE) consisting of a fracturable six-input LUT and two FFs. Within a half CLB (4 *subSlices*), only one clock (CLK) and three controls (Ctrls) are available to the FFs. Meanwhile, LUT data inputs in a BLE are restricted to a six-input LUT or two smaller LUTs with a maximum of five total inputs. Although the Ultrascale architecture contains carry chains and LUTRAMs, *elfPlace* does not handle them as they are not present in the ISPD'2016 contest benchmarks [2].

2) Stratix-IV: A Stratix-IV-like architecture consists of IOs and phase-locked loops (PLLs) sites along the periphery, with other resource types, such as *Slices*, DSPs, and memory blocks arranged in columns [34]. A *Slice* or a logic array block (LAB) contains 10 *subSlices* or adaptive logic modules (ALMs) consisting of a fracturable six-input LUT, two adders, and two FFs. The LAB FFs can choose from the 2-clock and 7-control signals available in the *Slice*. An ALM can be occupied by a single six-input LUT or two smaller LUTs, and the LUTs and FFs in an ALM are subject to a maximum total shared input count of eight. The considered Stratix-IV-like architecture also allows all *Slices* or LABs to be configured as memory blocks referred to as MLABs.

III. Hybrid Integrated Framework

Fig. 1 illustrates the proposed integrated placement flow, where an external placer can be integrated to *VTR*, highlighted in **brown**, in contrast to *VTR*'s typical flow in gray.

Section III-A describes the preprocessing capability of *VPR*'s external placement interface to generate inputs in bookshelf format [2] for *aug-elfPlace*. Section III-B details the enhancements in *aug-elfPlace* to target the Titan23 [25] benchmarks on a Stratix-IV-like architecture [34]. Section III-C details how the flat placement solution from *aug-elfPlace* is verified to ensure legality, enabling *VPR*'s $place$ stage to improve upon *aug-elfPlace*'s clustering, followed by routing and validation in *VPR*.

A. *Preprocess the Architecture and Netlist*

The ISPD'16 benchmarks [2] model most cluster types (e.g., IO, DSP, RAM) as simple tile-sized blocks, and so *elfPlace* cannot model all block types at the primitive level. *VPR* pre-clusters the more complex block types (e.g. DSP, IO, and memory) and leaves the common primitive types (e.g., LUTs, FFs) for *elfPlace* to place individually. An additional bookshelf input $'.lc'$ file is included to provide architecture-specific information such as legality constraints, instance and site dimensions, and a *VPR*-compatible flat placement output format. As the end user generally determines IO locations to meet board-level constraints, *VPR* also generates fixed IO placement that is required by *elfPlace* and most analytical placers.

B. *Enhanced aug-elfPlace*

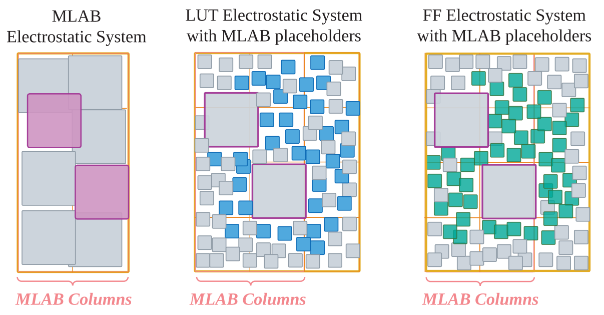
aug-elfPlace consists of an electrostatics-based flat global placer and a flat packer-legalizer without a detailed placer as the ISM-based algorithm in [8] cannot be applied to an Intel Stratix-IV [34] architecture due to the differences in the LUT/FF legality constraints.

1) Large DSP and Memory Blocks: While the min-cost flow approach in [8] can legalize the <1000 DSP/memory blocks in the synthetic ISPD'2016 contest benchmarks [2], it does not scale for the Titan23 benchmarks with >1000 DSP/memory blocks due to the limited available sites in the custom FPGA sitemaps. The potential sites for all the instances are considered iteratively based on the instance-to-site distance. Nevertheless, for the Titan23 benchmarks with similar site and instance count, a min-cost flow approach is forced to consider all possible instance-to-site matchings, resulting in a very large problem size for >1000 DSP/memory instances. Thus, we employ the scalable *auction* algorithm [35] in *aug-elfPlace* to legalize the large DSP and memory blocks.

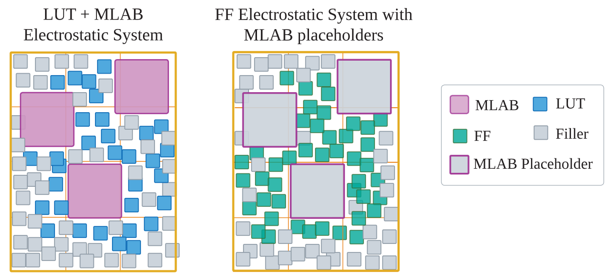
The auction algorithm solves an assignment problem by repeating the bidding and assignment phases [35] and has been used in ASIC detailed placement algorithm [36] due to its parallel and scalable nature. To assign N instances to M sites with $M \geq N$,

$$\begin{aligned} &minimize \sum_{i,j} c_{ij}\alpha_{ij}, \\ &subject\ to \sum_{i=1}^{N}\sum_{j=1}^{M} \alpha_{ij} = 1, \end{aligned} \tag{3}$$

where $\alpha_{ij} = 1$ if instance i at location (x_i, y_i) is assigned to site j at (x_j, y_j) and 0 otherwise, and $c_{ij} = abs(x_i - x_j) + abs(y_i - y_j)$ is the associated weight that depends on the distance between i and j. The second condition enforces $\alpha_{ij} = 1$ to ensure that only one instance is assigned to a site. During the bidding phase, each instance i bids for site j by



(a) Handling MLABs: Only specific LAB sites can be used for MLABs



(b) Handling MLABs: All LAB sites can be used for MLABs

Fig. 2. MLAB placement with placeholder fillers.

assigning a weight c_{ij}, and the lowest instance bidder for a site j is assigned during the assignment phase. The auction algorithm repeats the bidding and assignment phases until all the large blocks are legalized at their respective site locations.

2) MLAB Instances: In the memory-LAB or 'MLAB' mode, a LAB *Slice* can be used as a memory block. While LUTs, FFs, and adders can be placed on basic LAB and MLAB sites, MLABs are restricted to MLAB-specific sites. In *aug-elfPlace*, to ensure LUTs and FFs do not overlap with MLAB instances during global placement, we employ MLAB placeholder fillers that occupy the exact locations as the MLAB instances in the LUT/FF electrostatic systems, as illustrated in Fig. 2(a). The considered Intel Stratix-IV-like architecture [34] does not differentiate between LAB/MLAB sites and allows MLABs to be placed in any *Slice*/LAB site, allowing for handling in Fig. 2(b). This approach can also be used to handle LUTRAMs.

At the end of the global placement, MLABs are legalized using the min-cost flow approach in [8], as the number of available *Slice* sites is significantly larger than the MLABs.

3) Carry Chains: A *Slice*/LAB in the Intel Stratix-IV-like architecture can accommodate a maximum of 20 adders [34], and longer carry chains span multiple LAB sites. *aug-elfPlace* evaluates two approaches to handle carry chains during global placement, as follows:

a) ***Net Weighting****:* The net weighting approach assigns larger weights to carry chain nets to keep adder instances close to each other, thereby reducing their displacement during legalization. We consider equal net weights in both the x- and y-directions and higher weights along the x-direction, referred to as directional net weighting. The HPWL term in $W_e(x,y)$ in Eq. (1) is updated as,

$$W_e(x,y)_{\vec{wt}} = \sum_{e\in E}\Big(\omega_{ccX}(e)\max_{i,j\in e}|x_i - x_j| + \omega_{ccY}(e)\max_{i,j\in e}|y_i - y_j|\Big) \tag{4}$$

$$\omega_{ccX/Y}(e) = \begin{cases} w_{tX/Y}, & \text{if } e \text{ is a carry chain net} \\ 1, & \text{otherwise} \end{cases}$$

The x- and y-direction weights, $w_{tX} = \tau \times w_{tY}$, where $w_{tY} \in [1,10]$ as large values could negatively impact the convergence of the placer. Equal net weighting sets $\tau = 1$, while directed net weighting employs $\tau = 1.5$ (set empirically).

b) ***Partial Macro Representation****:* A macro representation has been employed for carry chains and large memory/DSP chains [37], [38] to maintain the relative ordering of the instances. In contrast, *aug-elfPlace* uses a 'partial macro' representation for carry chains, where a macro representation is employed for carry chains only during global placement, and the macros are broken down into individual adders for efficient packing and legalization.

For the carry chain 'partial macro' representation, an additional bookshelf input file ($'.cc'$) is included as an input to *aug-elfPlace* that lists the instances in each carry chain with relative ordering. After global placement, the carry chain instances are legalized greedily in decreasing order of chain lengths at the centroid location.

4) Ensuring LUT/FF Legality: While the legality constraints of LUTs and FFs are independent in the Ultrascale architecture, the Stratix-IV architecture has multiple legality constraints that involve both LUTs and FFs, as listed in Table I.

TABLE I
COMPARISON OF LUT/FF LEGALITY CONSTRAINTS.

FPGA Architecture	Legality Constraints		
	only LUTs	**only FFs**	**LUTs & FFs**
Ultrascale [20]	Max Shared Inputs	Clk and Ctrl signals	-
Stratix-IV [34]	-	Shared Ctrl signals	Max Shared Inputs + Restricted Loopbacks

a) ***Clustering-aware LUT/FF Area Update****:* During global placement, the areas of LUT/FF instances are inflated to ensure routability and ease of packing during legalization [8]. As the Stratix-IV-like architecture [34] enforces legality constraints that consider both LUTs and FFs, we inflate LUTs with higher inputs in *aug-elfPlace* to spread them out and limit FF instance area updates only based on clock signal sharing with neighboring FFs.

b) ***LUT/FF Packing & Legalization****:* The maximum shared input count in an ALM restricts the number of LUTs and FFs packed in an ALM. The packing-legalization runtime for Stratix-IV-like architecture is significantly higher than that of Ultrascale-like architecture, as all the existing instances in a LAB need to be checked when adding a new instance.

Thus, to prioritize the packing of larger LUTs in the Stratix-IV architecture, we update the cluster candidate score $\Omega(c, s)$ in [32],

$$\Omega(c, s) = \sum_{e \in E_c} \frac{InternalPins(e, c)}{TotalPins(e)} - \gamma.\Delta HPWL(c, s) \quad (5)$$

where E_c refers to all the nets of the cluster candidate instances, $TotalPins(e)$ is the total pins of a cluster net e, $InternalPins(e, c)$ is the total net pins of cluster instances that remain within c, and $\Delta HPWL(c, s)$ is the increase in HPWL to legalize cluster c at s. The updated cluster candidate score $\Omega(c, s)_{upd}$ in *aug-elfPlace*,

$$\begin{aligned} \Omega(c, s)_{upd} &= \Omega(c, s) + \beta.lut\_score(c) \\ lut\_score(c) &= \sum_{i \in c} \mathcal{L}.(|P_i^{IN}| - 1) \end{aligned} \quad (6)$$

where *lut_score(c)* determines the total input pin count of all LUTs in the cluster c, $\mathcal{L} = 1$ if instance i is a LUT and $\mathcal{L} = 0$ for a FF, and $|P_i^{IN}|$ refers to input pin count of instance i. The normalizing factor γ and the weighting factor β are empirically set to 0.02 and 0.1, respectively.

Although *aug-elfPlace* can generate legal clusters and placements for the Titan23 designs, some designs had < 5 legality failures related to the mode of operation of the instances (not considered in *aug-elfPlace*). These mode-related legality failures were identified and fixed by the VPR legalizer [23].

C. VPR: Legalize, Refine, and Evaluate

As shown in Fig. 1, the *VPR* legalizer ingests a flat placement solution from *aug-elfPlace* and generates inputs to *VPR*'s $placement$ and $routing$ stages [23]. The legalizer reconstructs and verifies each implicit input cluster, correcting legality and mode errors and using intra-cluster routing to check cluster legality [30]. It produces *VPR*-compatible $clustered\ netlist$ and $clustered\ placement$ files and diagnostic information, which helped recognize and address failure patterns in *aug-elfPlace*'s legality rules.

The legalized clustered placement is passed to the *VPR* placer as a starting point for simulated annealing optimization. After placement, *VPR* routes the design and reports wire length and critical path delay.

IV. Experimental Results

To demonstrate the effectiveness of the proposed open-source[1] integrated framework, the Titan23 [25] benchmarks targeting a Intel Stratix-IV-like architecture [34] were employed. *aug-elfPlace* is built on the open-source wirelength-driven flat analytical FPGA placer [9]. We employ *VPR*'s open-source place and route tool[2] referred to as '*VPR*$^{8+}$' with several placer and router enhancements over *VPR*8 [21]. All experiments were run on a Linux machine with a 64-core AMD EPYC 9554P CPU.

[1]*aug-elfPlace* is available at https://github.com/rachelselinar/aug-elfPlace; The *VPR* Legalizer is part of the *VTR* framework.

[2]Based on VTR github commit eb3c95d

TABLE II
Composition of the 'Titan23' Benchmarks [25].

Design	#IO / #PLL	#ALUT	#FF	#MLAB	#DSP	#M9K / #M144K	#CC*	#Insts	FPGA Layout
gaussianblur	558 / 0	804k	1054k	0	2	12 / 0	694	1858.7k	400 x 296
bitcoin_miner	385 / 1	444.2k	595.1k	0	0	1331 / 0	7,182	1041k	225 x 167
directrf	319 / 0	442.2k	447k	25,213	240	2535 / 0	22,863	917.5k	317 x 235
spartT1_chip2	1891 / 0	351.3k	392.5k	0	3	506 / 0	281	746.1k	280 x 107
LU_Network	404 / 2	186.1k	399.6k	8,173	112	1175 / 0	4,595	595.5k	220 x 163
LU230	373 / 0	208.6k	293.2k	0	116	5040 / 16	4,039	507.4k	430 x 319
mes_noc	5 / 8	272.8k	249.1k	0	0	800 / 0	2,185	522.6k	193 x 143
gsm_switch	136 / 1	159.1k	295k	0	0	1848 / 0	2,048	456.k	255 x 189
denoise	852 / 0	252.4k	9.7k	0	24	359 / 0	4,069	263.3k	150 x 111
sparcT2_core	451 / 0	173.4k	117.6k	0	0	260 / 0	103	291.7k	153 x 113
cholesky_bdti	162 / 0	75.7k	173.4k	331	132	600 / 0	2,184	250.3k	169 x 125
minres	229 / 1	112.8k	126.1k	0	78	1459 / 0	2,751	240.6k	225 x 167
stap_qrd	150 / 0	62k	161.8k	7,088	75	553 / 0	1,511	231.7k	158 x 117
openCV	208 / 0	107.8k	86.5k	0	213	785 / 40	1,828	195.6k	209 x 155
dart	69 / 0	103.8k	87.4k	0	0	530 / 0	2,061	191.8k	138 x 102
bitonic_mesh	119 / 0	108.6k	49.6k	0	85	1664 / 0	676	160k	242 x 179
segmentation	441 / 0	124.3k	7.5k	0	15	481 / 0	2,146	132.7k	136 x 101
SLAM_spheric	479 / 0	92.2k	18.3k	0	37	0 / 0	1,319	111k	95 x 70
des90	117 / 0	63.5k	30.2k	0	44	860 / 0	352	94.8k	171 x 127
cholesky_mc	262 / 0	28.6k	74k	767	59	444 / 16	884	104.2k	125 x 93
stereo_vision	506 / 0	38.2k	51.3k	0	76	113 / 0	2,298	90.2k	129 x 96
sparcT1_core	310 / 0	42.5k	44.9k	0	1	128 / 0	22	87.9k	82 x 61
neuron	77 / 0	24.1k	59.8k	0	89	136 / 0	616	84.3k	129 x 96

* Denotes actual number of carry chains (#CC) in the design.

A. Experimental Setup

1) Benchmarks: Table. II lists the 23 complex, non-synthetic designs in the Titan23 benchmark suite [25] containing adaptive LUTs (ALUTs), FFs, DSPs, two variants of block memories: M9K and M144K, along with IOs and PLLs, as listed in Table. II. An ALUT consists of a LUT and a one-bit adder synthesized as a single instance. The total instance count across the designs varies from $84k$ to $1.86M$, and 22% of ALUT primitives have carry chain adders [39]. The architecture represents large M144K and small M9K memory blocks as 1×8 and 1×1 *Slice* units, respectively, with DSP blocks as 1×4 *Slice* units. Each design uses an *auto-sized* FPGA that is chosen by *VPR* to be the minimal-size device that can accommodate all the design instances of various types while matching the relative number of memory, DSP, logic, and other columns in Stratix-IV devices.

2) Discussion on Baselines: This work considers *VPR*$^{8+}$ as the baseline with a default effort $inner\_num = 0.5$ that can operate in wirelength-driven (*VPR*$^{8+}$-WL) or timing-driven (*VPR*$^{8+}$-T) optimization modes.

HCAS: With an analytical global placement stage, legalization, and a simulated annealing refinement stage, HCAS [26] places the clustered blocks from *VPR*'s packer achieving 3% lower routed WL and 5% lower CPD with $10\times$ faster place runtime than *VPR*7+. HCAS's experiments were limited to 7 small designs ($< 31k$ logic blocks), and compared to *VPR*7+, *VPR*8 achieves 33% lower routed WL and 11% lower CPD with $\approx 2\times$ faster router runtime [21].

StarPlace: Using the clustered netlist from *VPR*'s packer, StarPlace employs a near-linear net model *star+* to place the blocks analytically and achieves $\approx 9\%$ lower CPD with $5\times$ faster place runtime than *VPR* on the small MCNC benchmarks [40]. Though the GPU-accelerated version of StarPlace achieves 3% lower routed WL and 26% lower CPD with $77\times$ faster place runtime than a wirelength-driven *VPR*7 on 7 IWLS benchmarks and an ARM7 benchmark [41], there are no published results on the Titan23 benchmarks available for comparison with *VPR*$^{8+}$.

Range-Based SA: RBSA employs a range-based algorithm to speed up *VPR*'s simulated annealer (SA) placer and achieves $2.5\times$ faster place runtime than *VPR*7 with similar quality of results [42]. ARBSA improves upon RBSA to achieve 10%

lower routed WL, 1% lower CPD with 2.82× faster place runtime than *VPR*7+ on 14 of the Titan23 benchmarks [43]. However, compared to *VPR*$^{8+}$-T, ARBSA achieves 53.6% higher routed WL and 7.9% higher CPD on the considered 14 Titan23 benchmarks.

LIQUID: By analytically placing the clustered blocks from *VPR*'s packer, LIQUID achieves similar CPD and routed WL with 23.7× faster place runtime than *VPR*7 on 18 of the Titan23 benchmarks [44]. With hierarchical force-based spreading, [45] improves upon LIQUID, achieving similar CPD and 8% lower routed WL with 15× faster place runtime than *VPR*7 on 11 of the Titan23 benchmarks. While LIQUID and its variants may be much faster, *VPR*$^{8+}$-T outperforms [45] with 22.5% lower routed WL and 48% lower CPD across the considered Titan23 benchmarks.

Packing & Analytical Placement: By improving both the pack and place stages, [46] achieves 50% lower placement HPWL with 18.3× faster pack-place runtime than *VPR*7 on 18 of the Titan23 benchmarks. [47] improves upon [46] with 9% lower CPD and 5% lower routed WL on 6 of the Titan23 benchmarks, with routing and validation by Quartus II. Comparing [47] routed by Quartus II to *VPR*$^{8+}$ post-route results, [47] achieves 28.3% lower routed WL with 77.8% higher CPD on the 6 considered Titan23 benchmarks.

Other Analytical Placers: Analytical placers such as [5]–[7], [9], [11], [17], [37], do not target an Intel Stratix-IV-like architecture and are not considered in this work.

3) VPR Settings: The default effort for *VPR* placer is set to $inner\_num = 0.5$. To enable wirelength-driven optimization in *VPR* placer, `--place_algorithm bounding_box` was used. All *VPR* router runs used: $route\_chan\_width = 300$, $max\_router\_iterations = 400$, and $astar\_fac = 1$.

4) Metrics: For all the placement flows, the metrics considered are placement bounding box wirelength estimate (PBB), Routed wirelength (RWL), and Post-Route Geomean critical path delay (CPD). The CPD reported is the geomean of all the longest paths within each clock domain, and routing in *VPR* was always run on timing-driven mode.

5) aug-elfPlace Carry Chain Handling: Fig. 3 compares the impact of the different carry chain handling schemes of *aug-elfPlace* on the overall placement HPWL across the Titan23 benchmarks (Section III-B3). The directional net weighting scheme with $w_y = 3$ outperforms the equal net weighting scheme with up to ≈ 1.5% lower overall placement HPWL. Regardless, the 'partial macro' representation achieves ≈ 2.5% lower overall placement HPWL due to minimal displacement of carry chain instances during legalization. Thus, *aug-elfPlace* employs a 'partial macro' representation for carry chains.

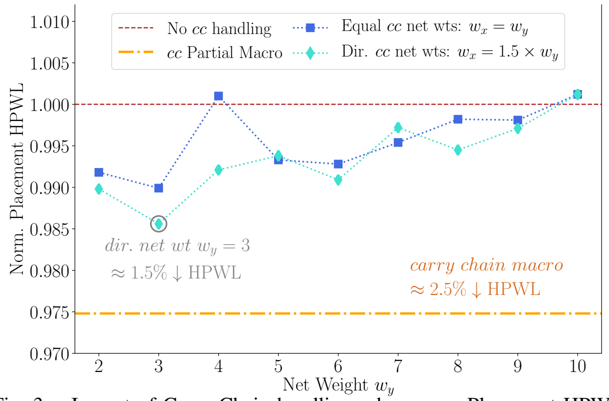


Fig. 3. Impact of Carry Chain handling schemes on Placement HPWL.

B. Integrated Framework (IF)

The integrated framework (IF) comprises the flat *aug-elfPlace* placer, *VPR* place & route tool along with *VPR*'s external placement interface consisting of the *VPR* legalizer [23]. As shown in Fig. 1, *VPR* generates bookshelf files for the Titan23 benchmarks that are used by *aug-elfPlace*. The flat placement solution from *aug-elfPlace* is verified, and mode-mismatches in instance locations are fixed by the *VPR* legalizer [23]. The interface generates a clustered netlist to allow *aug-elfPlace*'s placement to be further refined in *VPR*, followed by routing and validation.

1) No Refine (NR): The placement solutions from *aug-elfPlace* ingested through *VPR*'s legalizer are directly routed in *VPR* in the 'No Refine' (NR) flow without any placement refinements. Table III compares PBB, RWL and CPD of the NR flow with the timing-driven *VPR*$^{8+}$-T and wirelength-driven *VPR*$^{8+}$-WL baselines. Only one design *sparcT2_core* failed routing due to congestion in the NR flow, while the rest were successfully routed by *VPR*. Compared to *VPR*$^{8+}$-WL, the NR flow achieved 24%, 20%, and 15% improvements for PBB, RWL, and CPD, respectively, across all the Titan23 benchmarks. Against *VPR*$^{8+}$-T, the NR flow obtained 30% and 25% improvements for PBB and RWL, with 9% larger post-route geomean CPD across all the Titan23 benchmarks.

2) VPR SA Refinement: Two settings were employed to refine *aug-elfPlace*'s placement in *VPR*'s simulated annealer to target wirelength and timing optimizations.

- *Quench*: The input placement is refined at zero temperature by only accepting good moves to ensure the placement quality does not degrade.
- *Anneal*: Default settings for *VPR*'s annealer was employed where *aug-elfPlace*'s placement was used as the initial placement solution, and the annealer starts at a relatively low temperature (> 0) and accepts random moves to escape the local minima.

The refinement experiments are tabulated in Table III as *Quench-WL* and *Anneal-WL* for wirelength optimizations, and *Quench-T* and *Anneal-T* for timing optimizations. All the *Quench* and *Anneal* variants make *sparcT2_core* routable.

Wirelength Refinement: All refinement flows were run with the default effort of $inner\_num = 0.5$. *Quench-WL* improves NR flow with 2% lower PBB and RWL at the cost of a 4% increase in CPD, and is ≈ 20× faster than *Anneal-WL* across all the Titan23 benchmarks. *Anneal-WL* achieves the best wirelength improvement with 3% lower PBB and RWL than NR flow with a 1% lower CPD.

Timing Refinement: As the annealer only allows good moves that do not degrade the placement quality in *Quench* mode, *Quench-T* improves CPD and RWL of the NR flow

TABLE III
Placement BB WL estimate (PBB) and Routed WL (RWL) ($\times 10^5$) Comparisons with Post-Route Geomean CPD (ns).

Design	*VPR*$^{8+}$-T			*VPR*$^{8+}$-WL			Integrated Framework (IF): *aug-elfPlace* + *VPR*$^{8+}$														
							No Refine (NR)			*Quench-WL* (Q-WL)*			*Anneal-WL* (A-WL)*			*Quench-T* (Q-T)*			*Anneal-T* (A-T)*		
	PBB	RWL	CPD	PBB	RWL	CPD	PBB	RWL	CPD	PBB	RWL	CPD	PBB	RWL	CPD	PBB	RWL	CPD	PBB	RWL	CPD
gaussianblur	414.4	343.8	819.6	422.9	340.7	854.5	270.3	**204.5**	782.1	**262.1**	207.8	773.2	286.3	231.0	791.4	270.6	214.7	**762.9**	308.7	248.5	782.4
bitcoin_miner	90.8	104.0	**8.6**	82.7	95.5	12.1	76.3	92.8	10.5	75.7	91.9	11.4	**75.3**	**91.5**	10.6	77.2	92.8	11.4	83.3	101.5	10.2
directrf	132.3	139.1	8.9	109.9	115.3	13.2	78.2	89.4	8.1	75.3	85.6	8.0	**74.6**	**84.3**	9.3	76.1	84.9	8.0	102.9	112.1	**7.8**
sparcT1_chip2	66.2	77.2	**5.2**	62.9	73.6	6.4	42.2	51.9	5.7	41.7	51.3	5.9	**40.9**	**50.3**	6.0	43.8	52.6	5.6	58.9	65.3	6.2
LU_Network	48.4	58.1	5.2	49.7	59.9	6.2	42.0	54.8	5.5	41.1	53.9	5.7	**40.1**	**52.8**	5.6	42.1	54.6	5.3	48.1	60.6	**5.1**
LU230	172.4	180.8	10.6	162.5	169.4	18.5	87.9	91.7	19.1	**85.8**	**89.4**	17.2	87.8	91.4	15.0	87.5	90.0	17.3	144.0	149.5	**10.5**
mes_noc	41.6	51.3	**8.6**	41.4	50.8	10.0	31.9	42.2	9.5	31.7	41.9	9.5	**31.5**	**41.6**	9.4	32.3	42.2	9.2	49.6	61.0	9.2
gsm_switch	45.5	53.1	6.2	42.3	50.5	9.7	38.6	44.6	6.5	38.0	**44.1**	6.6	**37.5**	**44.1**	6.4	39.3	45.4	6.4	53.6	62.0	**5.7**
denoise	24.2	30.1	861.5	22.6	30.3	839.3	18.2	25.3	765.4	17.9	24.9	765.2	**17.7**	**24.7**	**761.8**	18.4	24.9	770.3	20.3	26.9	780.5
sparcT2_core	38.2	48.2	**11.0**	37.7	46.2	17.1	28.0	**FAIL**	**FAIL**	27.5	35.9	23.2	**26.6**	**34.4**	14.6	29.1	36.6	18.7	29.8	38.6	12.3
cholesky_bdti	20.1	26.3	9.1	19.1	24.8	10.4	14.1	19.1	8.3	13.8	18.7	8.4	**13.6**	**18.3**	8.5	14.3	18.8	8.0	14.9	19.8	**7.5**
minres	21.9	28.1	6.5	21.1	27.4	7.2	16.9	23.1	9.5	16.3	**22.2**	10.0	**16.2**	22.3	7.6	16.6	22.3	7.2	22.2	28.6	**6.0**
stap_qrd	24.1	28.3	**6.6**	21.2	26.1	8.0	17.4	22.5	8.3	16.8	21.9	8.2	**16.2**	**21.1**	8.3	17.3	22.3	7.6	20.4	25.4	7.1
openCV	27.4	34.2	**10.1**	24.8	30.8	19.3	17.5	23.7	17.8	16.9	22.8	17.6	**16.5**	**22.3**	17.5	17.4	23.0	16.9	27.3	33.4	12.8
dart	19.6	22.6	**14.7**	17.1	19.8	16.0	11.8	14.2	15.5	11.6	13.9	15.5	**11.5**	**13.7**	15.4	11.9	14.1	15.6	15.1	17.4	16.1
bitonic_mesh	39.3	49.8	14.5	35.4	45.7	19.0	21.4	33.4	11.8	20.4	31.5	11.5	**19.3**	**29.8**	11.0	20.6	31.2	11.1	20.7	30.9	**10.9**
segmentation	12.9	16.7	839.4	11.8	16.5	829.1	11.2	16.1	773.0	10.8	15.7	765.2	**10.4**	15.1	**758.8**	11.2	15.4	762.0	11.1	**15.0**	767.9
SLAM_spheric	12.7	16.3	78.4	11.5	15.0	84.0	8.4	11.6	74.3	**8.3**	**11.4**	74.2	**8.3**	11.5	74.6	**8.3**	**11.4**	74.1	8.6	11.8	**73.8**
des90	16.5	23.0	12.4	16.0	22.1	14.8	11.1	18.1	11.8	10.5	17.2	11.6	**9.8**	**15.9**	11.2	10.6	17.1	11.1	10.4	16.4	**10.8**
cholesky_mc	8.7	12.0	7.2	7.8	11.3	7.8	6.4	9.5	7.4	6.2	9.3	7.5	**6.1**	**9.1**	7.3	6.5	9.5	**6.6**	7.2	10.2	6.7
stereo_vision	5.1	5.8	**3.2**	5.0	5.7	3.3	5.4	6.4	3.7	5.2	6.0	3.6	**5.0**	**5.7**	3.7	5.3	6.0	3.8	5.3	6.0	3.6
sparcT1_core	9.1	12.6	**7.9**	9.3	12.9	9.2	6.4	9.2	9.6	6.4	9.2	9.6	**6.3**	**9.1**	9.6	6.6	9.6	9.8	6.9	10.0	9.5
neuron	6.2	7.9	**5.8**	6.0	7.6	7.2	4.6	6.3	5.9	4.4	6.0	6.2	**4.3**	**5.9**	5.9	4.5	6.0	5.9	4.9	6.4	6.5
Geo. Ratio	1.06	1.05	**1.00**	1.00	1.00	1.24	0.76	0.80	1.09	0.74	0.78	1.13	**0.73**	**0.77**	1.08	0.76	0.79	1.08	0.89	0.91	**1.00**

* All Quench and Anneal refinements in *VPR*$^{8+}$ were run with default effort $inner\_num = 0.5$.

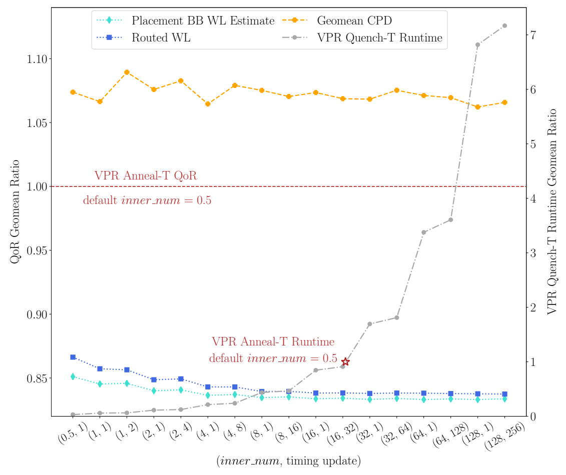


Fig. 4. Impact of *Quench-T* refinements on QoR and runtime for different values of $inner\_num$ and timing updates.

by 1%. On the other hand, *Anneal-T* significantly improves CPD of NR flow by 9% with a 11% increase in the RWL. Thus, *Anneal-T* refinement on *aug-elfPlace*'s placement achieves similar CPD at 14% lower routed wirelength compared to *VPR*$^{8+}$-T baseline across all the Titan23 benchmarks.

3) High-effort Quench-T Refinements: As *Quench-T* refinement with default effort $inner\_num = 0.5$ only improves the post-route geomean critical path delay by 1%, we evaluate the impact of higher effort levels ($inner\_num$) across all the Titan23 benchmarks in Fig. 4. *VPR* annealer's effort level $inner\_num$ is increased from 1 to 128 in multiples of two, along with an increased number of timing updates to allow for accurate delay calculations. As the effort level $inner\_num$ increases, the runtime increases substantially (in gray), and for $inner\_num = 16$ with 32 timing updates, the *Quench-T* runtime is the same as that of *Anneal-T* with default effort $inner\_num = 0.5$. The wirelength metrics PBB and RWL saturate with $< 2\%$ change for the increasing effort levels, while the timing metric CPD does not change much

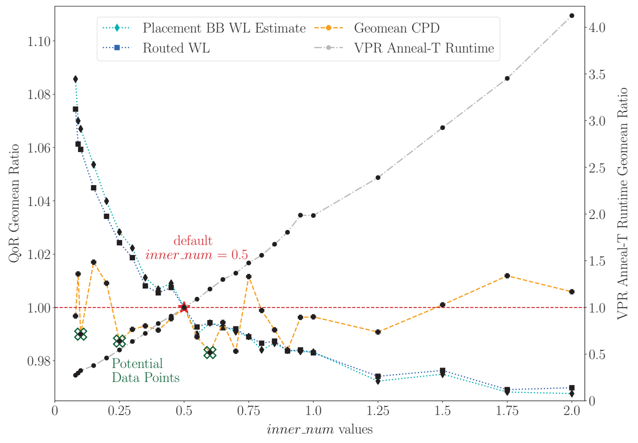


Fig. 5. Impact of *Anneal-T* refinements on QoR and runtime for different $inner\_num$ values.

regardless of the high effort levels. Thus, *Quench-T* cannot improve the CPD of a placement optimized for wirelength by *aug-elfPlace*.

4) High-effort Anneal-T Refinements: As *Anneal-T* refinement with default effort substantially improved *aug-elfPlace*'s placement quality to achieve similar CPD as *VPR*$^{8+}$-T, we assess the impact of *Anneal-T* with different effort levels ($inner\_num$) on QoR and runtime as depicted in Fig. 5. For very low effort levels $inner\_num \leq 0.08$, the two largest designs *gaussianblur* and *bitcoin_miner* have routing failures, and so the effort level $inner\_num$ was varied from 0.09 to 2.0. *Anneal-T* quickly improves CPD for low effort levels $inner\_num < 0.1$ at the cost of significant wirelength degradation. As effort level increases, *Anneal-T* focuses on improving wirelength while preserving the CPD improvements. The runtime increases with the effort level, and $inner\_num = 2.0$ takes $4.2\times$ longer than the default effort.

We identify three modes in Fig. 5 with varying effort levels: *Anneal-T* with $inner\_num = 0.1$ (*A-T0p1*), *Anneal-T* with $inner\_num = 0.25$ (*A-T0p25*), and *Anneal-T* with $inner\_num = 0.6$ (*A-T0p6*), that achieve lower CPD than the default effort mode.

5) Integrated Framework Placement Flows: The following eight different placement flows in the integrated framework (IF) are compared against the two baseline *VPR* flows:

- *IF:NR* or 'No Refine' — without refinements,
- *IF:Q-WL* — *Quench-WL* with default $inner\_num = 0.5$,
- *IF:A-WL* — *Anneal-WL* with default $inner\_num = 0.5$,
- *IF:Q-T* — *Quench-T* with default $inner\_num = 0.5$,
- *IF:A-T* — *Anneal-T* with default $inner\_num = 0.5$,
- *IF:A-T0p1* — *Anneal-T* with $inner\_num = 0.1$,
- *IF:A-T0p25* — *Anneal-T* with $inner\_num = 0.25$, and
- *IF:A-T0p6* — *Anneal-T* with $inner\_num = 0.6$.

QoR Comparisons: The placement bounding box wirelength estimate (PBB), the routed wirelength (RWL), and the post-route geomean critical path delay (CPD) for the different placement flows are compared in Fig. 6, across all the Titan23 benchmarks.

Compared to the baseline *VPR-WL* flow, *IF:NR* obtains 24% lower PBB, 20% lower RWL and 15% lower CPD. While *IF:Q-WL* achieves 2% lower PBB and RWL than $IF:NR$ with 4% higher CPD, *IF:A-WL* flow achieves the lowest WL metrics outperforming the baseline *VPR-WL* flow with 27% lower PBB, 23% lower RWL and 16% lower CPD.

IF:A-T achieves similar CPD as the *VPR-T* baseline with 17% lower PBB and 15% lower RWL. The *Anneal-T* variants *IF:A-T0p1* and *IF:A-T0p25* trade off $3-6\%$ PBB and $2-5\%$ RWL to achieve 1% lower CPD with lower *VPR* refinement runtime than *IF:A-T*. *IF:A-T0p6* achieves the best timing metrics with 2% lower CPD, 17% lower PBB and 15% lower RWL, compared to the baseline *VPR-T* flow.

Runtime Comparisons: The overall runtime for the baseline and IF placement flows, accounting for packing, placement, and routing for all the Titan23 benchmarks, is plotted in Fig. 7. The runtime for the baseline *VPR-T* and *VPR-WL* flows comprise $pack$, $place$, and $route$ runtimes. The overall placement and routing runtime for *VPR-T* is 1.8× higher than that of *VPR-WL* baseline flow. For the integrated framework placement flows, the overall runtime consists of time taken by *aug-elfPlace* placer, *VPR* legalizer [23], placement refinements in *VPR* annealer, and routing.It can be observed that the *VPR* placement runtimes of the *Quench* refinement flows are much faster than the *Anneal* refinement flows.

While all the *VPR* runs are on a single thread CPU, *aug-elfPlace* is run on 16 threads. Across the Titan23 benchmarks (geomean), a $1T$ *aug-elfPlace* placement is 1.2× faster than *VPR-T* $pack + place$, and a $16T$ *aug-elfPlace* is 5.9× faster. The *VPR* legalizer is 12% slower than the $16T$ *aug-elfPlace* placement runtime.

The wirelength reductions obtained by the IF placement flows result in significantly lower routing runtime than the baseline *VPR* flows. *VPR-T* routing is 1.34× slower than *VPR-WL* and 2× slower than $IF:NR$ flow. The increased routing runtime in the *VPR* baseline flows is mainly due to 3 large designs: *gaussianblur*, *directrf*, and *LU230*, that have ≈ 50× and ≈ 17× higher routing runtime for *VPR-T* and *VPR-WL* respectively, than *IF:NR*.

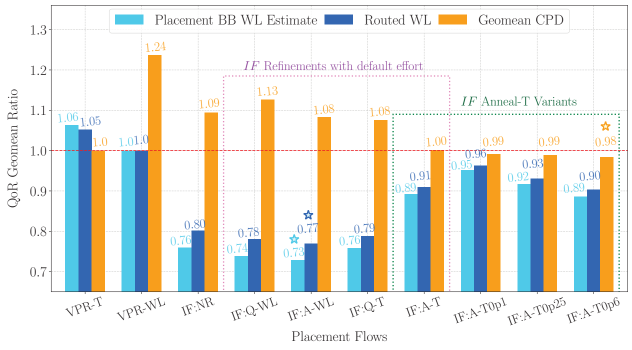


Fig. 6. QoR Comparisons of the different Placement Flows.

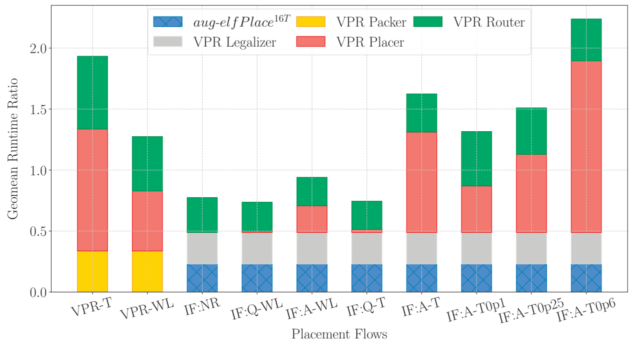


Fig. 7. Geomean Runtime Comparisons of the different Placement Flows.

All IF placement flows except *IF:A-T0p6* have lower overall placement runtime than the baseline *VPR-T* flow. The *IF:A-T0p6* flow takes 1.6× longer runtime for overall placement than the *VPR-T* baseline flow.

V. Conclusion

This work takes the first step to integrating a flat analytical placer into the flexible *VTR* framework to achieve the best of both worlds. We augment the analytical placer *elfPlace* [8] as *aug-elfPlace* with generalized architectural modeling to handle varied resource and site types and complex legality constraints. The *VPR* interface allows benchmarking and *VPR*'s legalizer [23] verifies legality of *aug-elfPlace*'s flat placement solution, followed by routing, validation, and refinement by *VPR*'s annealer. On average, across the Titan23 [25] benchmarks that target an Intel Stratix-IV-like architecture, the integrated framework achieves up to 2% lower geomean critical path delay with 15% lower routed wirelength, compared to *VPR* optimizing for timing metrics. As future work, we plan to incorporate timing awareness within *aug-elfPlace* to achieve even better quality of results.

Acknowledgment

This work was supported in part by Intel and VMWare through the Intel/VMWare Crossroads 3D-FPGA Academic Research Center. The authors thank Mohamed A. Elgammal (University of Toronto) for the valuable discussions on *VPR* clustering and Fatemehsadat Mahmoudi and Amin Mohaghegh (University of Toronto) for all the help setting up Quench and Anneal refinement experiments in *VPR*.

References

[1] C. Bobda, J. M. Mbongue, P. Chow, and et al., "The future of FPGA acceleration in datacenters and the cloud," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 15, no. 3, 2022.

[2] S. Yang, A. Gayasen, C. Mulpuri, S. Reddy, and R. Aggarwal, "Routability-driven FPGA placement contest," *Proc. of the International Symposium on Physical Design (ISPD)*, pp. 139–143, 2016.

[3] S. Yang, C. Mulpuri, S. Reddy, M. Kalase, S. Dasasathyan, M. E. Dehkordi, M. Tom, and R. Aggarwal, "Clock-aware FPGA placement contest," in *Proc. of the International Symposium on Physical Design (ISPD)*, 2017, p. 159–164.

[4] I. Bustany, G. Gasparyan, A. Gupta, A. B. Kahng, M. Kalase, W. Li, and B. Pramanik, "The 2023 MLCAD FPGA macro placement benchmark design suite and contest results," in *Proc. of the ACM/IEEE Workshop on Machine Learning for CAD (MLCAD)*, 2023.

[5] G. Chen, C.-W. Pui, W.-K. Chow, K.-C. Lam, J. Kuang, E. F. Y. Young, and B. Yu, "RippleFPGA: Routability-driven simultaneous packing and placement for modern FPGAs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 37, no. 10, pp. 2022–2035, 2018.

[6] Z. Abuowaimer, D. Maarouf, T. Martin, J. Foxcroft, G. Gréwal, S. Areibi, and A. Vannelli, "GPlace3. 0: Routability-driven analytic placer for ultrascale FPGA architectures," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 23, no. 5, pp. 1–33, 2018.

[7] W. Li, S. Dhar, and D. Z. Pan, "UTPlaceF: A routability-driven FPGA placer with physical and congestion aware packing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 37, no. 4, pp. 869–882, 2017.

[8] W. Li, Y. Lin, and D. Z. Pan, "elfPlace: Electrostatics-based placement for large-scale heterogeneous FPGAs," *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–8, 2019.

[9] R. S. Rajarathnam, M. B. Alawieh, Z. Jiang, M. Iyer, and D. Z. Pan, "DREAMPlaceFPGA: An open-source analytical placer for large scale heterogeneous FPGAs using deep-learning toolkit," *27th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 300–306, 2022.

[10] R. S. Rajarathnam, Z. Jiang, M. A. Iyer, and D. Z. Pan, "DREAMPlaceFPGA-PL: An open-source GPU-accelerated packer-legalizer for heterogeneous FPGAs," in *Proceedings of the International Symposium on Physical Design (ISPD)*, 2023, p. 175–184.

[11] J. Mai, J. Wang, Z. Di, G. Luo, Y. Liang, and Y. Lin, "OpenPARF: An open-source placement and routing framework for large-scale heterogeneous FPGAs with deep learning toolkit," in *IEEE 15th International Conference on ASIC (ASICON)*, 2023, pp. 1–4.

[12] S. Dhar and D. Z. Pan, "GDP: GPU accelerated detailed placement," *IEEE High Performance extreme Computing Conference (HPEC)*, pp. 1–7, 2018.

[13] S. Dhar, L. Singhal, M. A. Iyer, and D. Z. Pan, "FPGA accelerated FPGA placement," *29th International Conference on Field Programmable Logic and Applications (FPL)*, pp. 404–410, 2019.

[14] ——, "FPGA-accelerated spreading for global placement," *IEEE High Performance Extreme Computing Conference (HPEC)*, pp. 1–7, 2019.

[15] T. Martin, D. Maarouf, Z. Abuowaimer, A. Alhyari, G. Grewal, and S. Areibi, "A flat timing-driven placement flow for modern FPGAs," in *Proc. of the 56th Annual Design Automation Conference (DAC)*, 2019.

[16] J. Mai, J. Wang, Z. Di, and Y. Lin, "Multielectrostatic FPGA placement considering SLICEL–SLICEM heterogeneity, clock feasibility, and timing optimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 43, no. 2, pp. 641–653, 2024.

[17] T. Liang, G. Chen, J. Zhao, S. Sinha, and W. Zhang, "AMF-Placer 2.0: Open source timing-driven analytical mixed-size placer for large-scale heterogeneous FPGA," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 2024.

[18] Z. Lin, Y. Xie, G. Qian, J. Chen, S. Wang, J. Yu, and Y.-W. Chang, "Timing-driven placement for FPGAs with heterogeneous architectures and clock constraints," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2021, pp. 1564–1569.

[19] Z. Xiong, R. S. Rajarathnam, and D. Z. Pan, "A data-driven, congestion-aware and open-source timing-driven FPGA placer accelerated by GPUs," in *32nd International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2024.

[20] AMD, "Ultrascale architecture and product data sheet: Overview," Nov 2022. [Online]. Available: https://www.xilinx.com/content/dam/xilinx/support/documents/data_sheets/ds890-ultrascale-overview.pdf

[21] K. E. Murray, O. Petelin, S. Zhong, J. M. Wang, M. Eldafrawy, J.-P. Legault, and *et al.*, "VTR 8: High-performance CAD and customizable FPGA architecture modelling," *ACM Trans. Reconfigurable Technol. Syst. (TRETS)*, vol. 13, no. 2, May 2020.

[22] A. Arora, S. Mehta, V. Betz, and L. K. John, "Tensor slices to the rescue: Supercharging ML acceleration on FPGAs," in *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2021, p. 23–33.

[23] K. Thurmer and V. Betz, "VIPER: A VTR interface for placement with error resilience," in *Proc. of the 14th International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies (HEART)*, 2024, p. 99–108.

[24] X. Tang, E. Giacomin, A. Alacchi, B. Chauviere, and P.-E. Gaillardon, "OpenFPGA: An opensource framework enabling rapid prototyping of customizable FPGAs," *29th International Conference on Field Programmable Logic and Applications (FPL)*, pp. 367–374, 2019.

[25] K. E. Murray, S. Whitty, S. Liu, J. Luu, and V. Betz, "Titan: Enabling large and complex benchmarks in academic CAD," in *23rd International Conference on Field programmable Logic and Applications (FPL)*, 2013, pp. 1–8.

[26] C. Hu, Q. Duan, L. Hu, P. Lu, Z. Li, M. Yang, J. Wang, and J. Lai, "An analytical-based hybrid algorithm for FPGA placement," in *Proc. of the Great Lakes Symposium on VLSI (GLSVLSI)*, 2019, p. 351–354.

[27] S. Nikolić, F. Catthoor, Z. Tőkei, and P. Ienne, "Global is the new local: FPGA architecture at 5nm and beyond," in *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2021, p. 34–44.

[28] S. Srinivasan, A. Boutros, F. Mahmoudi, and V. Betz, "Placement optimization for NoC-enhanced FPGAs," in *IEEE 31st Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2023, pp. 41–51.

[29] K. E. Murray, M. A. Elgammal, V. Betz, T. Ansell, K. Rothman, and A. Comodi, "SymbiFlow and VPR: An open-source design flow for commercial and novel FPGAs," *IEEE Micro*, vol. 40, no. 4, p. 49–57, July 2020.

[30] J. Luu, J. Rose, and J. Anderson, "Towards interconnect-adaptive packing for FPGAs," in *Proc. of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2014, p. 21–30.

[31] M. A. Elgammal, K. E. Murray, and V. Betz, "RLPlace: Using reinforcement learning and smart perturbations to optimize FPGA placement," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 41, no. 8, pp. 2532–2545, 2022.

[32] 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, 2018.

[33] L. Singhal, M. A. Iyer, and S. Adya, "LSC: A large-scale consensus-based clustering algorithm for high-performance FPGAs," *54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2017.

[34] Intel, "Stratix-IV features," 2016. [Online]. Available: https://www.intel.com/content/www/us/en/content-details/654799/stratix-iv-device-handbook.html

[35] D. P. Bertsekas, "A new algorithm for the assignment problem," *Mathematical Programming*, vol. 21, no. 1, pp. 152–171, 1981.

[36] Y. Lin, W. Li, J. Gu, H. Ren, B. Khailany, and D. Z. Pan, "ABCDPlace: Accelerated batch-based concurrent detailed placement on multithreaded CPUs and GPUs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 39, no. 12, pp. 5083–5096, 2020.

[37] T. Liang, G. Chen, J. Zhao, S. Sinha, and W. Zhang, "AMF-Placer: High-performance analytical mixed-size placer for FPGA," in *IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2021, pp. 1–9.

[38] V. Betz and J. Rose, "VPR: a new packing, placement and routing tool for FPGA research," *Field-Programmable Logic and Applications, Springer*, pp. 213–222, 1997.

[39] K. E. Murray, J. Luu, M. J. P. Walker, C. McCullough, S. Wang, S. Huda, B. Yan, C. Chiasson, K. B. Kent, J. Anderson, J. Rose, and V. Betz, "Optimizing FPGA logic block architectures for arithmetic," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 6, pp. 1378–1391, 2020.

[40] M. Xu, G. Grewal, and S. Areibi, "StarPlace: A new analytic method for FPGA placement," *Integration*, vol. 44, no. 3, pp. 192–204, 2011.

[41] R. Pattison, C. Fobel, G. Grewal, and S. Areibi, "Scalable analytic placement for FPGA on GPGPU," in *International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, 2015, pp. 1–6.

[42] J. Yuan, L. Wang, X. Zhou, Y. Xia, and J. Hu, "RBSA: Range-based simulated annealing for FPGA placement," in *International Conference on Field Programmable Technology (ICFPT)*, 2017, pp. 1–8.

[43] J. Yuan, J. Chen, L. Wang, X. Zhou, Y. Xia, and J. Hu, "ARBSA: Adaptive range-based simulated annealing for FPGA placement," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 38, no. 12, pp. 2330–2342, 2019.

[44] D. Vercruyce, E. Vansteenkiste, and D. Stroobandt, "Liquid: High quality scalable placement for large heterogeneous FPGAs," in *International Conference on Field Programmable Technology (ICFPT)*, 2017, pp. 17–24.

[45] ——, "Hierarchical force-based block spreading for analytical FPGA placement," in *28th International Conference on Field Programmable Logic and Applications (FPL)*, 2018, pp. 26–263.

[46] Y.-C. Chen, S.-Y. Chen, and Y.-W. Chang, "Efficient and effective packing and analytical placement for large-scale heterogeneous FPGAs," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2014, pp. 647–654.

[47] S.-Y. Chen and Y.-W. Chang, "Routing-architecture-aware analytical placement for heterogeneous FPGAs," in *Proc. of the 52nd Annual Design Automation Conference (DAC)*, 2015.