AMF-Placer 2.0: Open Source Timing-driven Analytical Mixed-size Placer for Large-scale Heterogeneous FPGA

Tingyuan Liang, Gengjie Chen, Jieru Zhao, Sharad Sinha and Wei Zhang
[*]*ECE Department, Hong Kong University of Science and Technology;*
[†]*CSE Department, Chinese University of Hong Kong*
[‡]*CSE Department, Shanghai Jiao Tong University;* [§]*CSE Department, Indian Institute of Technology Goa*
tliang@connect.ust.hk, gjchen@cse.cuhk.edu.hk, zhao-jieru@sjtu.edu.cn, sharad@iitgoa.ac.in, eeweiz@ust.hk

Abstract*—Modern field-programmable gate arrays (FPGAs) may feature critical path portions of designs prearranged into movable macros during synthesis. These movable macros, with constraints of shape and resources, pose a challenge for mixed-size placement in FPGA designs that previous analytical placers cannot handle. Additionally, general timing-driven placement algorithms face challenges when dealing with real-world application designs and ultrascale FPGA architectures. To address these challenges, we present AMF-Placer 2.0, an open-source FPGA placer that supports mixed-size placement of heterogeneous resources. Building on AMF-Placer 1.0, AMF-Placer 2.0 incorporates new techniques for timing optimization, including an effective regression-based timing model, placement-blockage-aware anchor insertion, TNS/WNS-aware timing-driven quadratic placement, and sector-guided detailed placement. It is evaluated by a set of the latest large open-source benchmarks from various domains for AMD Xilinx Ultrascale FPGAs. Experimental results indicate that AMF-Placer 2.0 achieves critical path delays that are on average only 2.3% and 0.69% higher than those achieved by commercial tool AMD Xilinx Vivado 2020.2 and 2021.2, respectively. Furthermore, the average runtime of the placement procedure in AMF-Placer 2.0 is 7.0% and 11.5% lower than that of AMD Xilinx Vivado 2020.2 and 2021.2, respectively. Although limited by the absence of detailed information of devices and designs, AMF-Placer 2.0 is the first open-source FPGA placer that can handle timing-driven mixed-size placement for practical complex designs with various FPGA resources and achieve comparable quality to the latest commercial tools.

***Index Terms*—timing-driven placement, analytical placement, mixed-size placement, FPGA**

I. Introduction

An Field-Programmable Gate Array (FPGA) is a type of integrated circuit that can be reconfigured by users after manufacturing. The latest island-style FPGAs, such as the columnar and heterogeneous design shown in Fig.1, feature a 2D array of configurable sites, with each site consisting of basic elements of logic (BELs) [1]. Configurable logic block (CLB) sites, for example, are made up of BELs such as look-up tables (LUTs), flip-flops (FFs), multiplexers (MUXs), and carry chains (CARRYs). Other sites may contain larger, heterogeneous BELs, such as digital signal processors (DSPs) and block random access memories (BRAMs).

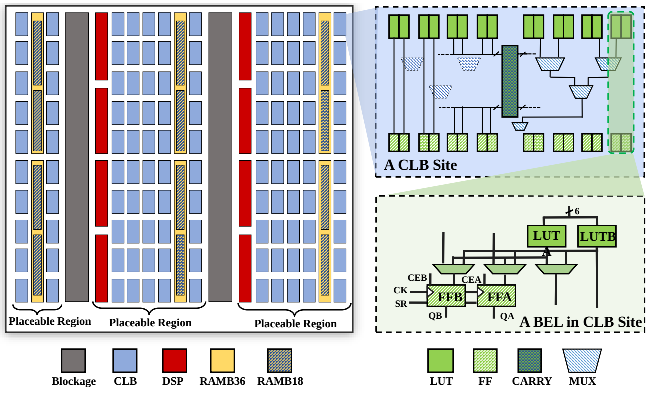


Fig. 1. Example of Xilinx Ultrascale FPGA device, a CLB site, and a BEL in it

During FPGA placement, the netlist generated by logic synthesis should be placed on discrete sites on the FPGA device. The goal is to realize shorter routing wirelength, less congestion regions and better timing, under the constraints of the device architecture. Typically, this placement flow includes the following steps: (1) initial placement/floorplanning to generates a very rough placement; (2) global placement to find optimal locations for the elements to optimize wirelength and timing under the resource constrains; (3) packing and legalization to exactly map each element to a valid FPGA site; (4) detailed placement to resolve the worst cases locally and optimize the metrics.

With advancement in semiconductor, FPGAs have increased in size as well as the variety of resources available on them and the overall architecture. Meanwhile, FPGA applications become much more complex and denser. These factors bring many new challenges to the placement flow.

Due to the upstream optimization, macros with constraints of shape and resource [2] [3] might be generated, like the examples shown in Fig. 2. On Ultrascale FPGA architecture [4], **standard cell** denotes the smallest, indivisible, representable instance, **occupying single BEL**, in the design netlist. Meanwhile, **macro** denotes a fixed group of multiple standard cells, **occupying multiple BELs**, with constraints of their relative locations. For example: (1) 1 MUX and 2 LUTs

Manuscript received XXXX; revised XXXX; accepted XXXX. Data of publication XXXX; Date of current version XXXX. This work was supported by the Hong Kong Research Grants Council General Research Fund under Grant 16213521. (Corresponding author: Jieru ZHAO)

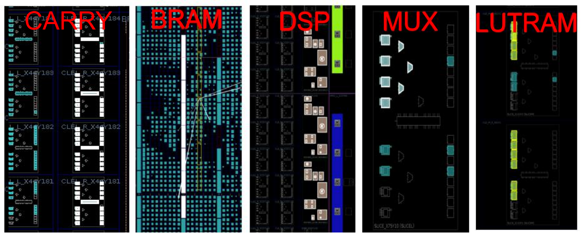


Fig. 2. Example of various types of macros with shape constraints: the macros are highlighted. They are combinations of logic blocks, **occupying multiple BELs**, with constraints of their relative locations.

connected to it should be treated as a macro; (2) a BRAM, without cascading with other BRAMs, is a standard cell; (3) 3 cascaded DSPs should be regarded as a macro. On the FPGA device, a macro might require multiple BELs spanning sites. Moreover, macros will lead to high interconnection density. These common scenarios in modern FPGA designs are seldom considered in previous exploration of FPGA placement.

A. *Related Works*

Some FPGA placers [5] [6], e.g., VTR, are based on simulated annealing (SA), which might lead to long placement runtime when the input netlist is large. Thus, analytical solutions using numerical approaches were proposed to solve the placement with high scalability and quality [7]. Gort and Anderson [7] presented an analytical FPGA placer HeAP, which demonstrated a 7.4× runtime advantage with 6% better placement quality compared to the SA placement algorithm of VPR 5.0. Chen et al. [8] proposed analytical placement solution with efficient and effective packing achieves 50% shorter wirelength, with an 18.30× overall speedup compared to VPR 7.0. During ISPD 2016/2017 contest, a series of analytical placers, e.g., UTPlaceF [9] [10] , RippleFPGA [11], GPlace [12] and NTUfplace [13], were inspired with the consideration of congestion and clock constraints and they showed promising performance on the contest benchmarks. Later in 2017, LIQUID [14] was proposed with analytical solution based on gradient-guided algorithm. ElfPlace [15] cast the placement density cost to the potential energy of an electrostatic system which tried to include various cost metrics in one nonlinear model to be optimized. However, the synthetic ISPD 2016/2017 benchmarks [16] [17] have some limitations. For example, the randomly-generated netlists contain impractical interconnections. Each 36Kb Block-RAM has only 6 input nets and 1 output net, and many registers are unnecessarily duplicated without consideration of timing and fanout. Furthermore, the netlists have no design hierarchy, which results in a relatively even distribution of timing criticality. Additionally, the benchmark lacks widely-used instances like CARRY, MUX, LUTRAM, and 18Kb Block-RAM. These limitations make the benchmark less representative of real-world scenarios.

Timing-driven FPGA placement is critical for achieving optimal timing quality in FPGA design implementation. However, this type of placement is challenging due to the complex interaction among routing wirelength, routability, and timing quality. Improper emphasis or neglect of the reduction of estimated wirelength, such as half perimeter wire length (HPWL), can result in routing failure, longer routing wirelength, and worse timing [18] [19]. Additionally, the discrete heterogeneous resource constraints of FPGAs pose significant challenges that are not accounted for in existing timing-driven solutions for ASIC placement [18], [20]–[27]. To address these challenges, researchers have proposed various solutions. Chen and Chang [28] proposed a local-routing-architecture-aware timing cost function in the analytical FPGA placement problem. Dhar, et al. [29] introduced an effective detailed placement based on the shortest path algorithm, which has been adopted by many solutions. Lin, et al. [30], [31] proposed an efficient delay model and timing-driven placement that consider clock constraints. Nikolić, et al. [32] developed an efficient ILP-based detailed placer that moves a carefully selected subset of LUTs to improve timing.

Real-world applications are becoming increasingly complex, and FPGA devices used to handle them have many architecture constraints. These constraints include placement blockages, packing legalization, clock legalization, and fixed macro shapes. As a result, timing-driven placement is becoming more difficult. Advanced algorithms and tools are needed to optimize placement while still meeting these complex restrictions. The existing challenges will be discussed in Section I-B.

B. *Motivation*

1) Challenges of FPGA Timing-driven Placement: In addition to the challenges we discussed in the introduction of AMF-Placer 1.0 [33] regarding mixed-size placement, there are still many unresolved challenges in timing-driven FPGA placement. These challenges span a wide range of perspectives and have not been addressed by previous works:

- Global placement algorithms in previous works [28], [30], [31] are mainly guided by TNS (total negative slacks), which is a scalar value and cannot capture the negative slack distribution among the nets. Neglecting the impact of WNS (worst negative slack) during global placement might result in suboptimal WNS results.
- The latest detailed placement algorithms are commonly based on the shortest path algorithm [29], which suffers from low-efficiency identification of candidate locations for involved instances.
- The complexity of application netlist and FPGA architectures has been raised dramatically. Complex FPGA architecture factors (e.g., placement blockage in Fig. 1) and mixed-size designs cannot be handled properly by the existing solutions [28], [30], [31].

2) Impact of Macro Instances on Timing Optimization : Mixed-size instances pose a significant challenge for timing optimization due to several factors.

- Large macros can cause disruptions in the placement of other instances in critical timing paths, even if the macros themselves are not part of those paths. This is because macros often require multiple sites or BELs. For instance, a CARRY macro can occupy over 128 BELs (including

LUTs, FFs, and CARRYs) and 8 CLB sites, leading to resource conflicts with many other instances.

- Macros, unlike standard cells, typically have a larger number of pins and nets connected to other instances. For example, the CARRY macro mentioned earlier may have over 300 nets connected to it outside of the macro. This is due to its inclusion of 64 LUT6 cells, each with 6 fanin pins. This high fanin and fanout of macros can lead to a significant number of intersections in critical timing paths, making timing optimization challenging. Moving an instance to optimize the timing of one path may result in a significant degradation of timing in many other paths.

C. Contributions

AMF-Placer 1.0 [33] enables efficient mixed-size FPGA placement with parallelized techniques including: (1) simulated-annealing-based floorplanning; (2) quadratic placement with interconnection-density-aware pseudo nets and legalization-oriented anchors; (3) cell spreading algorithm with utilization-guided search window and deadlock-free area supply control; (4) and progressive macro legalization. AMF-Placer 2.0 takes into account the practical demands of timing quality and the complexities of real-world applications with hierarchies that involve elements with shape constraints. It builds on the foundation laid by AMF-Placer 1.0 and offers several new essential features, including:

- a set of timing optimization algorithms that do not require static timing analysis (STA), such as path-length-aware SA-based floorplanning and parallelized CLB packing with timing factors considered.
- an efficient piecewise regression model of pin-to-pin delay, utilized by our integrated light-weight parallelized timing analysis engine.
- a placement-blockage-aware optimization scheme that identifies the potential negative interference of placement blockage with long paths. It spreads instances in specific regions and inserts placement anchors for the instances in the target paths to reduce cross-blockage routing.
- a WNS/TNS-aware timing-driven global placement algorithm based on quadratic programming and proper exploitation of pseudo-nets with slack-guided weights. This algorithm achieves multi-objective optimization of WNS, TNS, and wirelength.
- a sector-guided detailed placement algorithm that can efficiently identify instance movements with promising timing benefits.

The source code and Wiki of our proposed AMF-Placer 2.0 and involved open-source benchmarks are available at https://github.com/zslwyuan/AMF-Placer.

II. Preliminaries

In this section, we describe the mixed-size placement problem in FPGA scenarios and our analytical placement framework.

A. Characteristics of Mixed-size Design and Ultrascale Device Architecture

In Fig.2, we can see that standard cells in a macro must be placed in adjacent sites in the same column to meet downstream flow requirements. Typically, each macro includes one type of core cell, such as CARRY cells, MUX cells, LUTRAM cells, DSP cells, or BRAM cells. In addition to core cells, macros may also include peripheral LUT/FF cells directly connected to the core cells. Macro types can be primarily classified into five categories, each with distinct characteristics:

- The CARRYs connected with carry in/out port should be extracted as a macro, along with the LUTs and FFs directly connected to them. Furthermore, to enable the routing of some input pins of CARRY, which connect to signals outside the CLB site, some corresponding LUT slots in the same CLB site should be occupied by non-logic route-thru LUTs, which are not in the original netlist. Similarly, FF slots in CLBs may be unavailable due to routing resource contention in CARRY macros.
- A MUX with its two input standard cells, which could be two LUTs or two other MUXes, should be extracted as a macro. MUX macros may lead to route-thru usage of LUTs or disable external interconnection of some FFs due to the routing of selection signals.
- LUTRAM standard cells, which share input net for read-/write address and data bits, should be extracted as a macro. LUTRAM macros should be located in SLICEM columns of the device.
- For DSPs and RAMs, they might be cascaded to handle larger demand for computation and storage. The standard cells in one of these macros are interconnected by the nets of their cascaded input/output signals.

AMF-Placer 2.0 inherits the ability of AMF-Placer 1.0 [33] to detect the macros in the design netlist and generates placeholders to occupy resources and meet internal routing constraints. There are some other minor macros defined by vendor primitives [3], which are out of the scope of this work. More details are available in the device documentations [34]–[36].

In modern FPGA devices, placement blockages separate the device into several available placement regions, which may be introduced by IO banks (e.g., GPIOs and PCIe interfaces) or die boundaries, as shown in Fig.1. The delays of nets across the blockage region can be relatively higher than the delays of the nets routed within the general placement region. This problem is critical in timing-driven placement but is often ignored in existing works that focus on wirelength-driven placement. For example, the placement blockages of PCIe interfaces are omitted entirely in the device information of the benchmark ISPD 2016/2017 [16] [17].

B. Problem Formulation

The placement of the instances in a FPGA-based design can be formulated as a hypergraph $H = (V, E)$ placement problem. Let vertices $V = \{v_1, v_2, \ldots, v_n\}$ represent n instances in the design netlist and hyperedges $E = \{e_1, e_2, \ldots, e_m\}$

represent m nets. Let x_i and y_i be the x and y coordinates of the center of the instance v_i during placement, respectively. As mentioned in Section I, the instances can be categorized into two types, i.e., standard cells and macros, and both of these two types could be movable or fixed according to the design constraints. One of the most common objective functions for placement is the sum of half-perimeter wirelength (HPWL) over all nets, i.e., the defined E. By properly inserting weighted pin-to-pin pseudo nets, AMF-Placer 2.0 can integrate the timing objective into the conventional wirelength-driven placer. The mixed-size FPGA placer is responsible for determining the position of each movable instance (i.e., x_i and y_i) to minimize the objective function, which comprises wirelength and timing terms, while adhering to technology and region constraints.

C. The Framework of AMF-Placer 2.0

AMF-Placer 2.0 is a comprehensive FPGA placement framework as shown in Fig. 3. The input of AMF-Placer is the pre-implementation netlist extracted from AMD Xilinx Vivado. The outputs are the location of each instance on the specific device and a Tcl script for Vivado to consume the generated placement. The proposed placement flow consists of STA-independent phases and STA-dependent phases.

1) STA-independent Phases:

During the early stage of placement, instances can be moved extensively, making it challenging to obtain a reliable timing evaluation of the placement. As a result, STA-independent phases focus on early-stage timing optimization to minimize HPWL and pave the way for timing-driven phases. These STA-independent phases include:

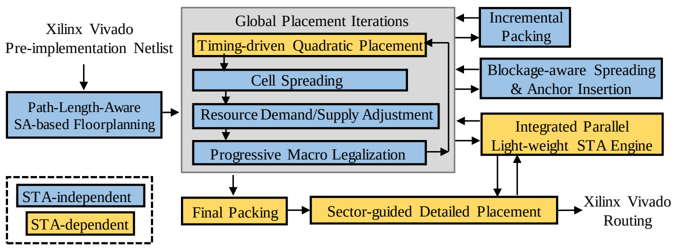
(1.1) Initial Floorplanning: AMF-Placer 2.0 begins with simulated-annealing(SA)-based floorplanning of the instance partitions obtained by the path-length-aware clustering and connectivity-based partitioning.

(1.2) Blockage-aware Spreading and Anchor Insertion: This phase analyzes the connectivity, the timing criticality of the circuits, and the location distribution of the critical paths on the device. It clusters some instances and inserts anchors for these instances to reduce the long cross-placement-blockage routing. To enable the coarse-grained movements of the clusters, instances in specific regions will be spread.

(1.3) Cell Spreading: Based on the area demand of instances and the area supply of the devices, instances will be spread from the regions where the demand for resources is outrunning supply, to other regions. We adopt the cell spreading algorithm of AMF-Placer 1.0 [33] for mixed-size instances.

(1.4) Resource Demand/Supply Adjustment: Based on the packing feasibility and routing congestion level, the area demands of instances and the area supply of some regions will be adjusted, to improve the placement quality. This phase is adopted from extended UTPlaceF [37].

(1.5) Progressive Macro Legalization: Mainly adopted from AMF-Placer 1.0 [33], each macro will be mapped to multiple potential locations or one exact location according to the confidence.



Xilinx Vivado Pre-implementation Netlist
Path-Length-Aware SA-based Floorplanning
Global Placement Iterations
Timing-driven Quadratic Placement
Cell Spreading
Resource Demand/Supply Adjustment
Progressive Macro Legalization
Incremental Packing
Blockage-aware Spreading & Anchor Insertion
Integrated Parallel Light-weight STA Engine
STA-independent
STA-dependent
Final Packing
Sector-guided Detailed Placement
Xilinx Vivado Routing

Fig. 3. The workflow of AMF-Placer 2.0 consisting STA-independent phases and STA-dependent phases

(1.6) Incremental Packing: Adopted from RippleFPGA [11], some LUTs and FFs will be paired as macros to improve the placement quality by identifying CLB internal nets.

2) STA-dependent Phases:

When the wirelength tends to be stable, timing-driven phases will further optimize the timing quality of the placement based on static timing analysis. These timing-driven phases include:

(2.1) Timing Model and Timing Analysis: Based on the dataset of timing delays, we use a piece-wise function based on non-integer polynomials to fit the distribution of timing delays. Based on this regression model of timing and the ideas of OpenTimer [38], a light-weight parallel static timing analysis engine is implemented.

(2.2) Timing-driven Quadratic Placement: To determine the next location of each instance, a quadratic optimization problem involving wirelength and timing is solved. In mixed-size placement, interconnection density and legalization are taken into account during quadratic placement. Additionally, timing-oriented pseudo nets are inserted between pins to optimize timing. The strength of these nets is determined by both the local timing slack of related paths and global timing quality.

(2.3) Global Packing: After the global placement iterations, the next phase involves exact legalization, which assigns each instance to specific sites with a fixed number and type of resources. For instance, BELs, including LUTs, FFs, MUXs, and CARRYs, are mapped to configurable logic blocks (CLBs). This phase focuses on maximizing the internal interconnection of CLBs while taking into account timing factors, under the constraints of available resources and clocking.

(2.4) Detailed Placement: Global placement and global packing evaluate placement quality from a global perspective, whereas detailed placement focuses on local critical paths that play a smaller role in global objectives. AMF-Placer 2.0 follows a similar placement flow to UTPlaceF [9] [10], RippleFPGA [11], and GPlace [12] by performing detailed placement after packing. This work flow can ensure legalization and identify available slots for instance replacement during detailed placement.

Most of the related algorithms for these phases are parallelized. Detailed methodologies will be illustrated in the following section III and IV.

III. Implementation of STA-independent Phases

In this section, we will illustrate the implementation of STA-independent phases which do not rely on static timing analysis (STA). Some STA-independent phases are based on

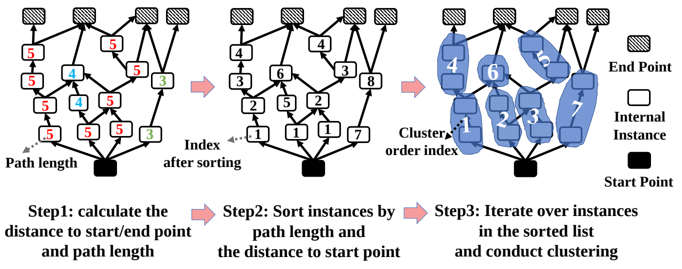

Path length
Index after sorting
Cluster order index
End Point
Internal Instance
Start Point
Step1: calculate the distance to start/end point and path length
Step2: Sort instances by path length and the distance to start point
Step3: Iterate over instances in the sorted list and conduct clustering

Fig. 4. Three steps of clustering guided by path length

AMF-Placer 1.0, and we will focus on the illustration of the additional important timing-oriented modifications.

A. Initial Floorplanning

Large-scale real-world FPGA designs, as shown in Section V, have specific architectures and hierarchies that differ from randomly generated FPGA netlists [16] [39] or small designs [7]. Additionally, FPGA resources are divided into discrete regions, and the supply of resources for each type is not uniform across the device. These design and device factors make the initial floorplan critical for later timing optimization, especially for large macros with high fan-in/fan-out.

Previous partitioning/clustering algorithms, such as those in [11] [33] [40] [41], do not account for timing factors during partitioning. Assigning weights to nets based on timing slack is a potential approach, but it may be impractical due to unreliable timing slack evaluation at the beginning of placement and the focus on minimizing total negative slack rather than worst negative slack. Besides, weighted hypergraph partitioning algorithms require additional operations, such as computing the sum of weights and sorting edges, which can increase complexity. Additionally, balancing weights across partitions further complicates the optimization problem.

To improve the timing optimization of AMF-Placer, version 2.0 introduces a novel clustering approach based on timing path length. By using breadth-first search (BFS) from the start and end points of timing paths, AMF-Placer 2.0 can easily obtain the distance of each instance to the farthest start or end point, resulting in the maximum length of the paths including that instance.

Next, AMF-Placer 2.0 sorts the instances according to their maximum path length (primary key) and the distance to the farthest start point (secondary key), generating a sorted list of instances. To focus on the instances in the longest paths, a path length threshold is introduced to select only the top 5% instances in the list.

AMF-Placer 2.0 iterates over instances in the sorted list, clustering each unclustered instance with its unclustered direct fanout instances as shown in Fig 4. The resulting fine-grained clusters are treated as entities during partitioning, reducing cross-cluster interconnections in critical paths.

AMF-Placer 2.0 does not directly cluster long paths. This is because an instance may be included in multiple paths with similar lengths, and clustering one path could negatively impact the timing of the other paths. Additionally, extremely long paths can still span a wide range in the final placement, and cluster them will limit later optimization. Instead, AMF-Placer 2.0 clusters instances with their direct fanout, realizing a more balanced and effective partitioning solution.

After the generation of fined-grained clusters, the conventional connectivity-based partitioning [40] will be involved to obtain tens of coarse-grained partitions of the design netlist. In this partitioning procedure, each of the fine-grained clusters are treated as an entity node. Finally, a simulated-annealing(SA)-based floorplanning in AMF-Placer 1.0 [33] will determine the location of partitions on the FPGA device.

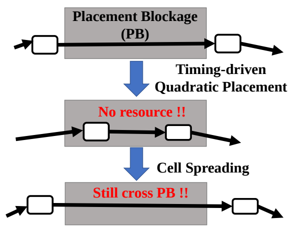
B. Blockage-aware Spreading and Anchor Insertion

Fig.5(a) demonstrates that simple timing-slack-based net weights cannot solve the timing problem caused by nets across a blockage region. This is because the reduced distance between instances across the blockage may be reverted by cell spreading or final packing. Furthermore, fine-grained local placement optimization is not effective in resolving this problem due to two main reasons. Firstly, one net spanning blockages may drive multiple sink instances, so resolving the problem for one sink instance may not address the cross-blockage problem for other sink instances, as shown by ① in Fig.5(b). Secondly, one net may belong to a long path. While resolving the problem of a pair of instances on the path, the cross-blockage problem for other instances in the path may arise, as shown by ② in Fig. 5(b).

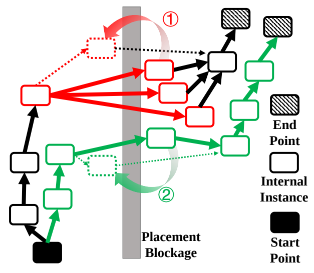
To address blockage-related problems during global placement, we propose a coarse-grained solution involving three steps as shown in Fig. 3.

First, critical path circuits are clustered based on a sorted list of instances with maximum path length greater than a threshold. Each instance is contained in only one of the resulting clusters, which are obtained by traversing unclustered direct/indirect successors of target instances in a depth-first search. The maximum size of each cluster is limited to an empirical number, which is set to 20000 in AMF-Placer 2.0.

Second, a target available placement region is selected for each cluster. If the proportion of instances belonging to a cluster in a specific available placement region is greater than 50%, that region is selected as the target region. Otherwise, the cluster is not assigned to any available placement region and instances in it will be released for the other clusters.



(a) Pure analytical solutions: cell spreading will spread instance from placement blockage and make the solution less effective.



(b) Fine-grained techniques: moving individual instances cannot effectively improve the critical path delay due to many long high-fanout paths.

Fig. 5. Drawbacks of the other timing optimization solutions for the columnar blockage regions

The third step involves guiding the movement of instances in a cluster to its corresponding target region during later placement iterations. In Ultrascale FPGA devices, available placement regions and blockages are typically columnar and horizontally aligned. Accordingly, each instance in the cluster is connected to a corresponding anchor at the horizontal center of the target region. The anchor has the same vertical coordinate as the instance. An example is shown in Fig. 6, where v_{Ai} is an anchor and v_i is an instance in the cluster. For the timing-driven quadratic placement which will be illustrated in Section IV-B, the weight of the pseudo net $e_b(v_i)$ connecting v_{Ai} and v_i can be formulated as:

$$w_{e_b}(v_i) = \quad \beta \times |(x_i - x_{Ai})| \times \text{pinNum}(v_i) \tag{1}$$

where β is a constant hyperparameter and $\text{pinNum}(v_i)$ is the number of the pins of v_i. For all the instances assigned to specific target regions, their blockage-aware pseudo nets will be recorded in a set $E_b = \{e_b(v_i)\}$. According to Eqn.(1), even for instances v_i in their target region, pseudo nets with lower weights will still be attached, and when v_i is far from v_{Ai}, the corresponding net will be strengthened.

An available target placement region could become highly utilized or congested during placement, so we need to stretch the placement in the target region. This is necessary to ensure that incoming clusters of instances have ample room for placement without causing serious congestion problems.

The formula for calculating the stretch ratio is: $\Delta r_{stretch} = N_{outside}/N_{inside}$, where $N_{outside}$ is the number of instances guided to the target region but not currently in it, and N_{inside} is the total number of instances in the target region that are not to be guided to other regions.

Let y_t be the vertical coordinate of the top instance v_t in the target region and y_b be the vertical coordinate of the bottom one. The linearly transformed vertical coordinates for v_t and v_b are calculated as follows:

$$y'_t = \quad y_t + \Delta r_{stretch} \times (y_t - y_b)/2 \tag{2}$$

$$y'_b = \quad y_b - \Delta r_{stretch} \times (y_t - y_b)/2 \tag{3}$$

Here, y'_t and y'_b are the new vertical coordinates for v_t and v_b, respectively.

Instances within the target region that have original vertical coordinates between y_t and y_b will be mapped to new locations using a linear transformation of coordinates, similar to the method shown in Fig. 6. In the case that y'_t or y'_b falls outside the boundary of the FPGA device, slight adjustments will ensure that all instances remain within the device boundary.

The insertion of anchors and specific instance spreading helps to gradually consume the clusters with long paths across blockages by mapping their instances to their corresponding placement regions. This approach ensures that available placement space is efficiently utilized, while also avoiding congestion problems.

IV. Implementation of STA-dependent Phases

A. Regression-based Timing Model and Timing Analysis

An accurate timing delay model is crucial in timing-driven placement. Some previous solutions for timing delay modeling [31] [42] [43] in FPGA placement are based on timing look-up table or ideal routing path. They may suffer from limitations in applicability and accuracy. Some solutions [31] [44] lack availability since they rely on industry-provided dataset.

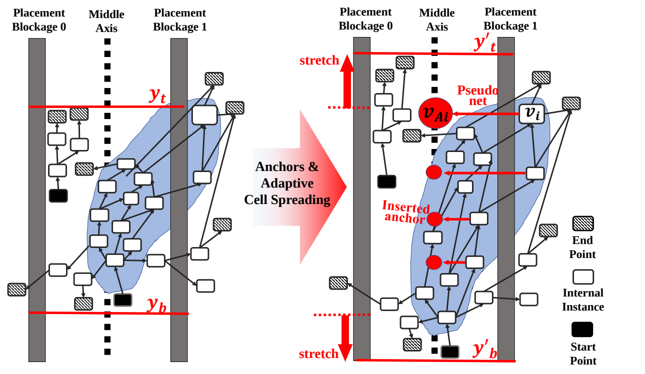


Fig. 6. Examples of Blockage-aware Spreading and Anchor Insertion: some of the pseudo nets are omitted for better visualization.

AMF-Placer 2.0 employs both a regression model and a look-up table. The fixed logic delay (T_{logic}) is recorded in the look-up table, while the regression model estimates the variable net delay (T_{net}). To create the regression model, we generate a dataset of instance-to-instance timing delays of nets in real-world benchmarks, as described in Section V-A. We randomly extract this data using the Vivado Tcl command ”get_net_delays” and visualize it in Fig.7. The dataset consists of N_{sample} samples that represent the locations of instances and the routing delay between them, as determined by the actual routing. In AMF-Placer2.0, we set N_{sample} to 10000, which is sufficient for the factors in the regression model to converge. We use a non-integer polynomial function to fit the distribution of timing delays, accounting for the differences in X-Y coordinates of interconnected instances on the FPGA. The formula of the net delay estimation function $T_{net}(e_{i,j})$ is:

$$\begin{aligned} T_{net}(e_{i,j}) =& a_0\Delta x^{b_0} + a_1\Delta x^{b_1} + a_2\Delta y^{b_0} + a_3\Delta y^{b_1} \\ & - Cas(e_{i,j}) + Bkg(x_i, x_j) \\ & with \quad \Delta x = |x_i - x_j|, \quad \Delta y = |y_i - y_j| \end{aligned} \tag{4}$$

It includes regression factors (a_0, a_1, a_2, a_3, b_0, b_1) and considers the distance between the instances (Δx, Δy), the delay deduction for cascaded standard cells in one macro ($Cas(e_{i,j})$), and the extra routing delay for instances in different placement regions ($Bkg(x_i, x_j)$). The values of $Cas(e_{i,j})$ and $Bkg(x_i, x_j)$ are determined by look-up tables. To improve estimation accuracy, the timing estimation function is divided into several concatenated intervals based on the Euclidean distance between instances ($D_{Eucl} = \|(\Delta x, \Delta y)\|$). The regression factors may be different among these intervals, making $T_{net}(e_{i,j})$ a piece-wise function. This approach accounts for the fact of imbalanced dataset that the number of long-routing nets is significantly smaller than the number of short-routing nets in real designs for sampling.

In the implementation of AMF-Placer 2.0 for UltraScale FPGAs (xcvu095), the timing delay model (T_{net}) is divided into three intervals of Euclidean distance (D_{Eucl}): $[0, 3)$, $[3, 6)$, and $[6, +\infty)$. The regression factors b_0 and b_1 are found to be 0.3 and 0.5, respectively. The resulting regression model is

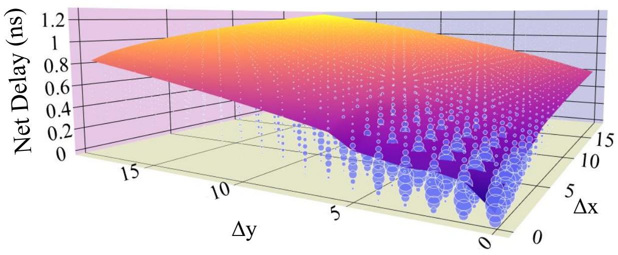


Fig. 7. Sample Distribution and Regression Model Surface: The markers of varying sizes depict the density of the samples in our dataset. The surface is generated by the piece-wise function T_{net} with different Δx and Δy

TABLE I
NET DELAY MODEL VALIDATION WITH BENCHMARKS

Benchmark Name	BLSTM	Rosetta DigitRecog	Rosetta FaceDetect	SpooNN	MemN2N	Minimap2	OpenPiton	Average
AMF CPD Prediction (ns)	7.23	9.62	18.37	9.34	9.62	7.61	11.23	-
|Relative Error| (%)	15.56	8.94	7.35	6.39	9.83	4.43	7.64	8.59
Vivado Pre-Route Prediction CPD (ns)	9.17	12.25	18.59	8.54	12.02	8.19	12.59	-
|Relative Error| (%)	7.13	16.00	6.25	2.69	12.61	2.93	3.52	7.29
Actual Post-Route CPD (ns)	8.56	10.56	19.83	8.78	10.67	7.96	12.16	-

visualized in Fig.7. The dataset shows a maximum delay of 3.469 ns and a minimum delay of 0 ns. The regression model based on these data has a mean error of 0.147 ns and a standard deviation of 0.225 ns. TableI shows that the average relative error of our predicted critical path delay (CPD) is 8.59%, compared to the Vivado post-route exact CPD. The average relative error of Vivado pre-route CPD estimation is 7.29%. Our proposed net delay model is relatively optimistic because most of the samples in the dataset are not from congested regions, while critical paths usually route through congested regions.

In AMF-Placer 2.0, a light-weight parallel static timing analysis (STA) engine has been implemented based on the timing model for T_{net} and T_{logic}, as well as the fundamental idea of OpenTimer [38]. The netlist is treated as a directed acyclic graph with topological levelization, and the timing analysis of instances at the same level is performed in parallel. For each instance v_i, the actual arrival time will be:

$$T_{arr}(v_i) = \max_{v_j \in fanin(v_i)} (T_{arr}(v_j) + T_{logic}(v_j) + T_{net}(e_{i,j})) \tag{5}$$

and the required arrival time will be:

$$T_{req}(v_i) = \min_{v_j \in fanout(v_i)} (T_{req}(v_j) - T_{net}(e_{i,j})) - T_{logic}(v_i) \tag{6}$$

Accordingly, similar to [44] [30], for later timing optimization, the slack of a timing edge $e_t(i,j)$ between source instance v_i and sink instance v_j is defined as:

$$\begin{aligned} Slack(e_t(i,j)) = & T_{req}(v_j) - T_{arr}(v_i) \\ & - T_{logic}(v_i) - T_{net}(e_{i,j}) \end{aligned} \tag{7}$$

We can get a set of timing edges with negative timing slack $E_t = \{e_t(i,j) | Slack(e_t(i,j)) < 0\}$.

B. Timing-driven Quadratic Placement

AMF-Placer 2.0's timing-driven placement strategy offers benefits in both local and global timing optimization. Locally, it adapts pseudo net weights based on timing slack values to resolve local timing violations. Globally, it uses a percentile-based evaluation of global timing quality and adjust the bias of the weights to improve timing at the design level.

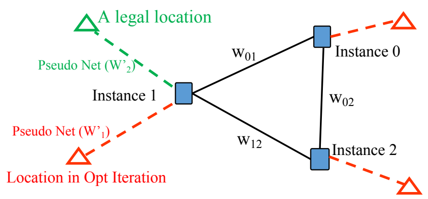


Fig. 8. An Example of Pseudo Nets and Anchors: the dash lines indicate pseudo nets and the triangles represent anchors.

1) Problem Modeling Overview: As mentioned in Section I-A, analytical placers approximate the wirelength (or HPWL) and some other metrics using numerical models for efficient solutions. Like many previous analytical placers [7] [9] [11] [37], to approximate the non-differentiable HPWL function, AMF-Placer uses weighted quadratic objective function $\widetilde{W_e}$, as follows:

$$\widetilde{W_e} = \sum_{i,j \in e} [w^{B2B}_{x,ij}(x_i - x_j)^2 + w^{B2B}_{y,ij}(y_i - y_j)^2], \tag{8}$$

where e is a net. Accordingly, i and j represent the instances connected to e. Meanwhile $w^{B2B}_{x,ij}$ and $w^{B2B}_{y,ij}$ are weights set according to Bound2Bound net model [45].

To realize timing-driven placement, AMF-Placer 2.0 formulate a minimization problem as follows:

$$\begin{aligned} \min_{\mathbf{x},\mathbf{y}} \quad & (1-\lambda) W_{WL}(\mathbf{x},\mathbf{y}) + \lambda W_T(\mathbf{x},\mathbf{y}) \\ \text{s.t.} \quad & W_{WL}(\mathbf{x},\mathbf{y}) = \sum_{e \in E} \widetilde{W_e} + \sum_{e_p \in E_p} [w_{e_p} \widetilde{W_{e_p}}] \\ & W_T(\mathbf{x},\mathbf{y}) = \sum_{e_b \in E_b} [w_{e_b} \widetilde{W_{e_b}}] + \sum_{e_t \in E_t} [w_{e_t} \widetilde{W_{e_t}}] \end{aligned} \tag{9}$$

In this context, apart from the design net in E, there are three types of additional pseudo nets as shown in Fig. 8.

- E_p represents the set of general pseudo nets used for legalization and density control [33], and w_{e_p} denotes the specific weight assigned to each pseudo net.
- E_b is a set of blockage-aware pseudo nets, as defined in Section III-B, and its specific weight, w_{e_b}, is defined in Equation (1).
- E_t is a set of timing-slack-aware pseudo nets, as defined in Section IV-A, and its weight, w_{e_t}, is related to the timing slack value of e_t, as specified in Equation (7).

$\widetilde{W_{e_b}}$, $\widetilde{W_{e_b}}$ and $\widetilde{W_{e_t}}$ are the HPWL quadratic approximation for the corresponding pseudo nets. Meanwhile, λ balances the trade-off between wirelength and timing during the placement process, gradually transitioning from wirelength-driven placement to timing-driven placement. This allows the algorithm to achieve better timing performance while minimizing wirelength and area overheads.

2) TNS/WNS-Aware Pseudo Nets: The pseudo nets in E_t are pin-to-pin interconnections, and their HPWL approximation ($\widetilde{W_{e_t}}$) is simply the Manhattan distance between the two connected pins. This distance provides a basic estimate of the timing delay associated with the interconnection. The goal of minimizing this term $\sum_{e_t \in E_t} [w_{e_t} \widetilde{W_{e_t}}]$ is to reduce the approximate total negative slack (TNS), which is the sum of the negative slack of all the timing paths. Since the timing edges in E_t could be located in timing paths with different negative slacks, more weights should be assigned to e_t in the most critical timing paths. In previous solutions [31] [44], the criticality of e_t was

$$w_{e_t} = \left(1 - \frac{Slack(e_t)}{Dly_{max}}\right)^{\alpha}$$

where Dly_{max} represents the target delay value, and α is the constant criticality exponent that makes the weights adaptive to the nets with different criticalities. The distribution of timing slack can vary across different applications or stages of placement. For example, some control logic instances or macros with high fanout may have many tiny negative slacks and only a few significantly worse ones. This can make it difficult for the analytical model to improve the worst negative slack. Additionally, in most cases, only a small portion of paths violate timing constraints during placement iterations, making it unnecessary to set timing-aware pin-to-pin nets for all interconnections in the circuit.

Considering these factors, in contrast, in AMF-Placer 2.0, the criticality of e_t is extended as follows:

$$w_{e_t} = \begin{cases} 0 & (Slack(e_t) \geq 0) \\ \left(1 - \dfrac{Slack(e_t)}{Dly_{max}}\right)^{C(Slack(e_t))} & (Slack(e_t) < 0) \end{cases} \tag{10}$$

with

$$C(Slack(e_t)) = max\left(\alpha, \frac{\beta Slack(e_t)}{T_{thr}}\right) \tag{11}$$

where the parameters α and β serve to restrict the possible values of the exponent, while T_{thr} represents a percentile value used to evaluate timing quality globally. Specifically, N_{thr} percent of edges exhibit negative timing slack worse than the threshold T_{thr}. This threshold is updated during STA before each quadratic placement iteration. Based on Eqn(11) the weight w_{e_t} will be set aggressively high for edges with significantly negative slack values exceeding T_{thr}. The self-adaptive exponent can address the WNS issue. For edges with less severe negative slack values better than T_{thr}, a relatively low weight value will be used. These small negative slack values will still be accumulated to minimize TNS. In practice, α, β, and N_{thr} are set to 0.9, 3, and 30, respectively.

Finally, the Eigen3 solver [46] is adopted to handle the optimization for wirelength, TNS and WNS in Eqn.(9), with the high parallelism based on OpenMP.

C. Global Packing

In the exact legalization after FPGA global placement, each instance must be mapped to a specific site on the FPGA, which contains a fixed number and type of resources. For example, LUTs, FFs, MUXes, and CARRYs with compatible signals can be grouped into a CLB site, reducing inter-site routing, as illustrated in Fig. 1.

A high-quality parallelized packing algorithm has been proposed by UTPlaceF [37], allowing FPGA sites to concurrently search for candidate packing solutions and negotiate during synchronization. AMF-Placer 1.0 incorporated modifications to improve efficiency in this packing solution. In AMF-Placer 2.0, a timing factor was added to the priority evaluation of instances during parallel packing to optimize timing.

In the site-centric parallel packing algorithm, instances with locations near a site compete to occupy its resources. In the original implementation of [37], a score function is introduced as follows:

$$\begin{aligned}\text{Score}_{UTP}\left(C, s_k\right) = &\sum_{e \in \text{Net}(C)} \frac{\text{InternalPins}\left(e, C\right) - 1}{\text{TotalPins}\left(e\right) - 1} \\ &- \theta \cdot \Delta\text{HPWL}\left(C, s_k\right)\end{aligned} \tag{12}$$

where

- C is a candidate cluster of instances
- Net(C) is the set of nets that have at least one cell in C
- TotalPins(e) is the total pin count of net e
- InternalPins(e,C) is the number of pins of net e in C
- $\Delta\text{HPWL}\left(C, s_k\right)$ is the HPWL increase of moving the instances in C from their global placement locations to site s_k
- θ is a positive weighting parameter.

The priority for instance v_i to be packed into the site s_k is determined by the increase of $\text{Score}_{UTP}\left(C, s_k\right)$ when inserting v_i into C.

In Equation (12), the first term aims to reduce inter-site routing, while the second term minimizes wirelength overhead. During this process, an instance v_i may be mapped to a distant site due to resource contention. After timing-driven global placement, critical path instances should be packed into sites close to their original positions to preserve timing results.

To achieve this, we utilize the longest length of paths including the instance as an extra metric during global packing. This is because placement variations in instances along a long path can accumulate and significantly degrade path delay, requiring significant effort during detailed placement to recover. For example, in benchmark FaceDetect [47], there are hundreds of timing paths with more than 50 instances. One path may have a delay of around 13ns after global placement, which may seem acceptable given the 15ns clock period constraint. However, if each instance in the path contributes a small delay increase of just 0.1ns, the path delay can dramatically increase to over 18ns, resulting in serious timing violations.

Accordingly, we include an extra term when evaluating a candidate cluster as follows:

$$\text{Score}_{AMF} = \text{Score}_{UTP} + \gamma \sum_{v_i \in C} \text{maxPathLen}(v_i) \tag{13}$$

where maxPathLen(v_i) is the longest length of paths including the instance and γ is a parameter which we set to 0.05 empirically while θ in Score_{UTP} is set to 0.01.

Apart from the timing optimization, clock legalization is implemented as AMF-Placer 1.0. Additionally, there are multiple BEL slots in one CLB site as shown in Fig. 1. Therefore, for the instances mapped to the same CLB site, they will be mapped to the BEL slots by enumerating possible mapping solutions to minimize the maximum delay of involved timing paths.

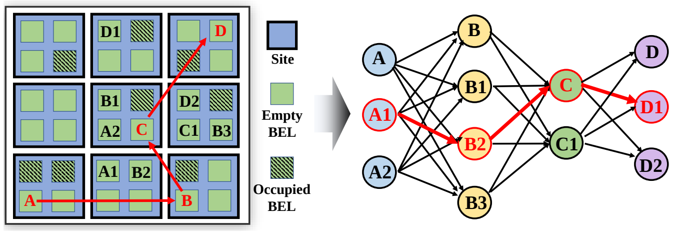


Fig. 9. An example of shortest-path-based detailed placement: A, B, C and D have 3,4,2 and 3 candidate sites, respectively.

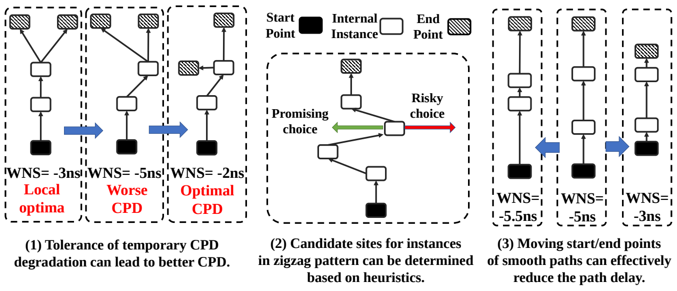

WNS= -3ns
Local optima
WNS= -5ns
Worse CPD
WNS= -2ns
Optimal CPD
(1) Tolerance of temporary CPD degradation can lead to better CPD.
Start Point
Internal Instance
End Point
Promising choice
Risky choice
(2) Candidate sites for instances in zigzag pattern can be determined based on heuristics.
WNS= -5.5ns
WNS= -5ns
WNS= -3ns
(3) Moving start/end points of smooth paths can effectively reduce the path delay.

Fig. 10. Examples explaining the motivation of detailed placement strategies

D. Detailed Placement

Global placement and packing can hardly reach optimal timing delays of critical paths due to resource contention and mapping instances to specific sites. To solve this problem and minimize the worst negative slack (WNS), the detailed placement process is conducted locally on critical paths while minimizing interference with other instances. To do this, AMF-Placer 2.0 adopts the widely-used shortest path algorithm [29], [44], [48], which identifies the best candidate locations for each node on a timing path under optimization. This algorithm determines the location for each node to minimize the total delay of the target critical path. This process is illustrated in Fig.9, where a various number of candidate locations are identified for each node on the critical path A->B->C->D. To solve this problem, a directed acyclic graph is constructed with multiple layers, each related to one instance in the critical path and containing vertices corresponding to candidate sites for the instance. Finding the shortest path from the source layer to the sink layer can then map the instances in the path to new sites and reduce the timing delay of this path. Compared to previous solutions, AMF-Placer 2.0 offers several key features. Firstly, it dynamically adjusts the scope of the target critical paths and the candidate sites for them, allowing for faster optimization with fewer iterations of incremental STA. Secondly, it tolerates a temporary increase in WNS during the shortest-path-based detailed placement, effectively resolving the problem related to local optima as shown in Fig. 10(1). Finally, it identifies promising candidates to reduce the number of candidate sites, reducing computational complexity and improving optimization.

1) Self-Adaptation of Optimization Scope and Tolerance of Temporary Timing Degradation: As shown in Algorithm 1, AMF-Placer 2.0 conducts N_{DPI} iterations of detailed placement, each processing N_{CP} of the critical paths with the highest delay sequentially. For each target path, a conventional shortest path algorithm is applied. The instances in the previously optimized critical paths are marked fixed to preserve optimization results during the detailed placement for the less critical paths. R_{nbr} controls the maximum distance between instances and their corresponding candidate sites, ranging from 1 to 0.1. At the beginning of each iteration, R_{nbr} is decreased by ΔR to reduce the number of candidate sites for each instance, and N_{CP} is increased by ΔN_{CP}. By controlling R_{nbr} and N_{CP}, more candidate sites and fewer target critical paths are used at the beginning of the detailed placement to realize fast improvement of CPD and WNS. In contrast, fewer candidate sites and more target critical paths are considered in the final iterations to fine-tune the locations of the instances and further improve timing optimization. The lowest CPD and its corresponding placement are recorded. If AMF-Placer 2.0 has not improved the CPD for I_{thr} iterations, it reverts to the best recorded placement. This involves increasing R_{nbr} by $(I_{thr}+1)\Delta R$ to provide more potential candidate sites for the most critical paths and reducing N_{CP} by $(I_{thr}+1)\Delta N_{CP}$ to focus on the most critical paths. In an interval of I_{thr} iterations, AMF-Placer 2.0 tolerates the increase of CPD of the circuit. Empirically, N_{DPI}, ΔN_{CP}, ΔR, and I_{thr} are set to 120, 20, 0.01, and 5, respectively.

2) Smart Candidate Identification: AMF-Placer 2.0 includes a sector-guided candidate selection algorithm for the concrete optimization of a specific path (i.e., line 8-9 in Algorithm 1). This algorithm identifies a smaller number of promising candidate sites for path delay optimization. Conventional solutions select candidate sites from a $d \times d$ square window around each instance on the path, which may be inefficient. AMF-Placer 2.0 makes two important observations

Algorithm 1: Detailed Placement of AMF-Placer 2.0

```
Input: netlist information $H = (V, E)$, device information
       $DI$, instance locations after global packing
       $\mathbf{P} = \{(x_i^{pk}, y_i^{pk})\}$
Output: instance locations after optimization
        $\mathbf{P}' = \{(x_i^{dp}, y_i^{dp})\}$
1  $\mathbf{P}'$ = $\mathbf{P}$
2  bestCPD = staticTimingAnalysis($H$,$DI$,$\mathbf{P}'$);
3  notOptCnt = 0; // recording the times of optimization failures
4  $N_{CP}$, $R_{nrb}$ = 1, 1.0;
5  for $i$=0; $i$< $N_{DPI}$; $i$++ do
6      $L_{CP}$ = getMostCriticalPaths($N_{CP}$);
7      foreach path $\in L_{CP}$ do
8          candMap = findCandidateLoc(path, $\mathbf{P}'$, $DI$, $R_{nrb}$);
9          $\mathbf{P}'$ = shortestPath(path, candMap, $\mathbf{P}'$);
10         markInstancesFixed(path);
11     markInstancesUnfixed($L_{CP}$);
12     CPD = staticTimingAnalysis($H$,$DI$,$\mathbf{P}'$);
13     $R_{nrb}$ -= $\Delta R$; $N_{CP}$ += $\Delta N_{CP}$; notOptCnt += 1;
14     if CPD<bestCPD then
15         bestCPD = CPD;
16         recordTheBest($\mathbf{P}'$);
17         notOptCnt = 0;
18     else if notOptCnt ==$I_{thr}$ then
19         recoverToBest($\mathbf{P}'$);
20         notOptCnt = 0;
21         $R_{nrb}$ += $(I_{thr}+1)\Delta R$;
22         $N_{CP}$ -= $(I_{thr}+1)\Delta N_{CP}$;
23 recoverToBest($\mathbf{P}'$);
24 optPerPathPerSingle_DowngradeNotAllowed($H$,$DI$,$\mathbf{P}'$);
25 return $\mathbf{P}'$
```

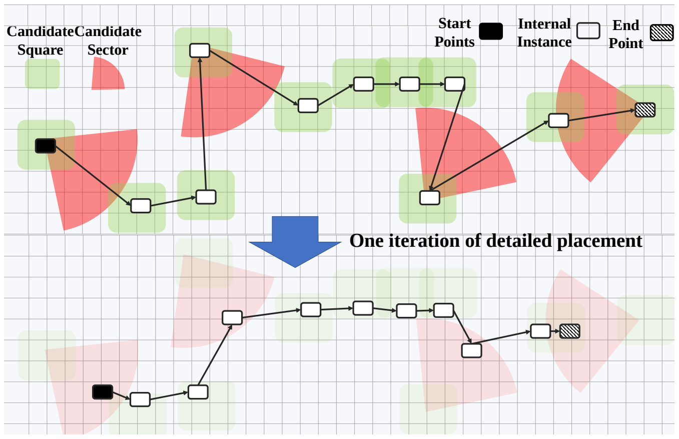


Fig. 11. Examples of sector-based candidate site window

based on the examples shown in Fig. 10: (1) for zigzag patterns in a path, beneficial candidate sites for an instance at a turning point can be identified based on its predecessor and successor; (2) for a smooth or straight path, its overall delay is mainly determined by its start/end points.

To identify candidates, AMF-Placer 2.0 uses small $d \times d$ square windows for all instances on the path, where $d = R_{nbr}R_{Square}$. Additionally, sector windows are set for specific instances, which could be start/end points or internal instances forming an acute triangle with their predecessor and successor. The center of a sector window overlaps with the corresponding instance and the radius is R_{Sector}, greater than R_{Square}. For internal instances, the direction of the window aligns with the angle bisector of the triangle formed by the instance, its predecessor, and its successor, with the instance serving as the vertex. For start/end points, the direction of the sector window aligns with the direction vector from the instance to its successor/predecessor. An example is shown in Fig. 11. Overlapping candidate windows may lead to resource/architecture conflicts. If some BEL slots in these involved CLB sites conflict between instances, the assignment of BEL slots is resolved by balancing the candidate supply among instances, similar to [29]. With this candidate selection algorithm, the runtime complexity can be significantly reduced. Empirically, R_{Square} and R_{Sector} are set to 3 and 5, respectively. In summary, the number of candidate sites for an instance is determined by both the global timing optimization progress and the local pattern of the instances on critical paths.

At the final stage of detailed placement, AMF-Placer 2.0 will enable the constraint of WNS, i.e., no CPD increase will be accepted, to further fine-tune the placement. For detailed placement, an integrated parallel incremental timing analysis is implemented to fast evaluate the timing benefit.

Previous works [29], [44], [48] on placement optimization may suffer from limited timing optimization and high runtime overhead due to two reasons. Firstly, they are constrained by the requirement that no moves or swaps should increase the WNS, which may lead to local optima and limit the optimization. Secondly, the runtime complexity of the shortest path-based solutions is O($pL_{avg}N^2_{candidate}$) [44], where p is a small constant overhead factor, L_{avg} is the average length of the critical paths considered, and $N_{candidate}$ is the average candidate site set size for critical path nodes. A small $N_{candidate}$ may limit the optimization space, while a large $N_{candidate}$ may significantly increase the runtime. In contrast, the solutions presented in AMF-Placer 2.0 overcome these limitations effectively.

TABLE II
PARAMETERS OF BENCHMARKS AND DEVICE

Benchmarks	Rosetta [47] FaceDetection	SpooNN [49]	MiniMap2 [56]	OpenPiton [52]	MemN2N [50]	BLSTM [51]	Rosetta [47] DigitRecog	Device: VU095
#LUT	68945	63095	407586	180388	184997	118967	151636	537600
#FF	56987	70987	252624	111966	84694	54690	105580	537600
#CARRY	4978	2091	19826	1712	11528	2762	1970	33600
#Mux	2177	217	180	13696	4466	36210	4662	201600
#LUTRAM	255	251	251	752	3500	1147	251	19200
#DSP	101	165	528	58	312	258	1	768
#BRAM	141	208	283	147	148	812	379	1728
#Cell	134450	137937	681889	309145	289721	215101	265775	-
#Macro	3582	1135	8746	8278	5775	14651	3061	-
#siteForMacro	55666	23079	191263	48066	118960	171822	55754	-
MacroRatio	40%	16%	28%	15%	41%	80%	21%	-
clockPeriod(ns)	15	8	8	10	10	8	8	-

V. Experimental Evaluation

A. Target Device, Benchmarks and Environment

AMF-Placer 2.0 targets AMD Xilinx VU095 FPGA devices and can transfer related techniques to other devices [29]. The benchmarks used in the experiments are open-source, suitable for VU095 devices, and designed for various domains, such as CNN [49], memory networks [50], LSTM [51], SoC/NoC [52], and genetic encode alignment [53]. Benchmark OptimSoC [54] used by AMF-Placer 1.0 is removed since its major cause of timing violation is clock domain crossing, which is outside the scope of AMF-Placer 2.0. Table II lists the parameters of the benchmarks and device, where macroRatio indicates the macro proportion of the design. Some of the benchmarks are generated via high-level synthesis [47], while others are described in Verilog. Commercial IP cores in some benchmarks are black-box instances in the post-synthesis netlist and cannot be handled by RapidWright [55] and other placers relying on the EDIF netlists. To handle general designs, AMF-Placer 2.0 directly extracts the instance interconnection from Vivado via interactive Tcl commands.

AMF-Placer 2.0 is implemented in C++ and experiments are conducted on Ubuntu 20.04 with an Intel i7-6770 CPU (3.40 GHz, 8 logic cores) and 32GB DDR4. All the experiments in this section use 8 threads.

B. Comparison with Vivado

Existing open-source analytical FPGA placers do not support mixed-size FPGA placement of the aforementioned macros on Ultrascale devices. Therefore, for a comprehensive comparison, we used the widely-used commercial tool Xilinx Vivado 2020.2 and 2021.2 as our baselines. Notably, Xilinx Vivado 2021.2 employs machine learning, which is interesting for comparison.

The experimental results are presented in Table III, where WNS represents the worst negative slack, CPD represents critical path delay, and RT represents the runtime of the entire placement flow. To evaluate the final timing quality, the placement results of AMF-Placer 2.0 were loaded by Vivado router to implement actual routing. Different combinations of

TABLE III
COMPARISON OF AMF-PLACER 2.0 WITH VIVADO 2020.2 AND 2021.2

	Place-Route	BLSTM	DigitRecog	FaceDetect	SpooNN	MemN2N	MiniMap2	OpenPiton	Average
WNS(ns)	AMF-V2020	-0.389	-3.595	-0.384	-0.491	-1.601	0.049	-2.310	-
	AMF-V2021	-0.508	-4.373	-0.445	-0.537	-1.665	0.001	-2.556	-
	V2020	-0.562	-2.564	-0.243	-0.779	-0.669	0.037	-2.159	-
	V2021	-0.668	-3.249	-0.264	-0.836	-0.732	0.070	-2.436	-
CPD(ns)	**AMF-V2020**	8.40	11.64	15.39	8.50	11.60	7.96	12.38	-
	Rnorm	**0.981**	1.097	1.009	**0.967**	1.087	**0.999**	1.017	1.023
	AMF-V2021	8.51	12.41	15.45	8.55	11.66	8.01	12.56	-
	Rnorm	0.994	1.169	1.013	0.972	1.093	1.006	1.032	1.040
	V2020	8.56	10.61	15.24	8.79	10.67	7.96	12.17	-
	Rnorm	1	**1**	**1**	1	**1**	1	**1**	1
	V2021	8.67	11.29	15.27	8.85	10.73	7.94	12.44	-
	Rnorm	1.012	1.065	1.002	1.006	1.006	**0.996**	1.022	1.016
RT(s)	AMF	337	544	245	205	748	920	565	-
	Rnorm	**0.85**	1.04	**0.86**	**0.77**	1.15	**0.84**	1.02	0.93
	V2020	398	522	285	265	650	1094	555	
	Rnorm	1.00	**1.00**	1.00	1.00	**1.00**	1.00	**1.00**	1.00
	V2021	396	529	300	271	759	1031	630	
	Rnorm	0.99	1.01	1.05	1.02	1.17	0.94	1.14	1.05

TABLE IV
EFFECTIVENESS OF PROPOSED OPTIMIZATION TECHNIQUES

	Configuration	BLSTM	DigitRecog	FaceDetect	SpooNN	MemN2N	MiniMap2	OpenPiton	Average
CPD(ns)	Cfg0	8.40	11.64	15.39	8.50	11.60	7.96	12.38	-
	Rnorm	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	1.000	1.000
	Cfg1	9.31	13.28	16.25	10.18	13.30	8.04	12.72	-
	Rnorm	1.108	1.141	1.056	1.197	1.146	1.011	1.028	1.098
	Cfg2	8.58	12.36	18.00	10.07	13.42	8.55	13.52	-
	Rnorm	1.021	1.062	1.170	1.185	1.157	1.075	1.092	1.109
	Cfg3	8.42	12.33	17.35	10.86	12.42	8.61	13.18	-
	Rnorm	1.003	1.060	1.127	1.278	1.071	1.082	1.065	1.098
	Cfg4	9.22	12.21	15.42	8.65	11.74	7.96	12.67	-
	Rnorm	1.098	1.049	1.002	1.018	1.012	1.001	1.024	1.029
	Cfg5	8.48	11.78	15.45	8.50	11.80	7.97	12.35	-
	Rnorm	1.010	1.012	1.004	1.000	1.017	1.001	**0.998**	1.006
	Cfg6	8.53	12.24	15.55	9.82	13.41	8.17	14.71	-
	Rnorm	1.016	1.052	1.010	1.155	1.156	1.027	1.189	1.086
RT(s)	Cfg0	337	544	245	205	748	920	565	-
	Rnorm	1.00	1.00	1.00	1.00	1.00	**1.00**	1.00	1.00
	Cfg1	326	528	323	189	743	986	550	-
	Rnorm	**0.97**	0.97	1.32	**0.92**	0.99	1.07	0.97	1.03
	Cfg2	346	524	186	190	505	937	510	-
	Rnorm	1.03	0.96	**0.76**	0.93	0.68	1.02	**0.90**	0.90
	Cfg3	351	483	279	196	445	1007	595	-
	Rnorm	1.04	**0.89**	1.14	0.95	**0.60**	1.09	1.05	0.97
	Cfg4	355	527	245	204	729	966	558	-
	Rnorm	1.05	0.97	1.00	1.00	0.97	1.05	0.99	1.00
	Cfg5	353	513	252	209	842	967	568	-
	Rnorm	1.05	0.94	1.03	1.02	1.12	1.05	1.00	1.03
	Cfg6	381	609	246	195	826	998	656	-
	Rnorm	1.13	1.12	1.01	0.95	1.10	1.09	1.16	1.08

placer and router may lead to different results, which is also evident in Table III.

Here, AMF represents AMF-Placer 2.0, V2020 represents Vivado 2020.2, and V2021 represents Vivado 2021.2. For example, AMF-V2020 represents the combination of AMF-Placer 2.0 and the router of Vivado 2020.1. The WNS metrics, which are negative or near-zero in Table III, indicate that the designer-defined timing constraints are strict for the placement. The CPD metrics mainly indicate the delay of the critical path. All the results of runtime and critical path delay are normalized to the results of Vivado 2020.2 for comparison, as indicated by the Rnorm values.

Based on the table, Vivado 2020.2 is currently the top-performing option in terms of both timing quality and runtime, and its router is particularly well-suited for use with AMF-Placer 2.0. However, it is worth noting that AMF-V2020 achieves only a 2.3% higher CPD and 11.5% lower runtime on average when compared to Vivado 2020.2, and a 0.69% higher CPD and 7.0% lower runtime on average when compared to Vivado 2021.2. It is important to keep in mind that the runtime comparison is only for reference, as AMF-Placer 2.0 skips certain post-placement optimizations such as LUT pin assignment and clock skew optimization. Nonetheless, AMF-Placer 2.0 is the first FPGA placer capable of handling timing-driven mixed-size placement of complex designs using various FPGA resources, and it achieves comparable quality to the latest commercial tools. According to the analysis of the concrete placement, we find that the major existing limitations of AMF-Placer 2.0 can be categorized into to two types, listed as follows:

- Timing estimation accuracy:As evaluated in Section IV-A, the timing estimation model used in AMF-Placer 2.0 is relatively optimistic, as it does not consider some congested regions that overlap with critical paths. This issue is particularly noticeable for benchmarks DigitRecog [47] and MemN2N [50], and may be addressed in the future by using machine-learning-based timing estimation approaches, such as [57]. Additionally, the timing analysis at the floorplanning stage may be insufficient, leading to a poor initial placement. Moreover, a significant portion of the WNS for benchmark DigitRecog [47] is caused by clock skew, which is not considered by existing works for the Ultrascale FPGA architecture. This problem may be resolved by using the approach proposed in [58].
- Design-aware factors: The design netlists used in Vivado placement are hierarchical in nature. The design hierarchy information can be used to fully leverage regularity for the placement of local circuits, such as parallel accumulator datapaths, buses, and FIFOs. However, AMF-Placer 2.0 uses a flattened netlist, which misses out on opportunities for regularity-related optimization. This limitation can be addressed by adopting approaches from existing works, such as [59] and [60].

C. Effectiveness of Proposed Optimization Techniques

In this subsection, we evaluate the individual impact of the major proposed optimizations for different placement phases in AMF-Placer 2.0. Here we show their impact by setting the placer configurations as follows, to disable different specified optimization technique:

- *Cfg0*: all the optimization techniques are enabled as the configuration in Section V-B.
- *Cfg1*: disable path-length-aware clustering before partitioning in Section III-A
- *Cfg2*: disable blockage-aware spreading and anchor insertion in Section III-B.
- *Cfg3*: disable WNS-aware timing criticality pseudo net weight in Section IV-B, i.e. $C(Slack(e_t)) = \alpha$
- *Cfg4*: disable path-length-aware parallel packing in Section IV-C
- *Cfg5*: disable sector-guided site candidate selection in Section IV-D and the value of R_{Square} remains at 3 (i.e., just using the original small square window)
- *Cfg6*: disable sector-guided site candidate selection in Section IV-D and set R_{Square} to be 5 (i.e., simply enlarge the square window)

As shown in Table III, it is evident that AMF-Placer 2.0 performs better when paired with the router of Vivado 2020.2. Therefore, we have chosen the ”AMF-V2020” configuration as baseline for the comparisons.

Table IV presents the results of the experiments conducted with different configurations, with all results normalized to those of *Cfg0* for ease of comparison, as indicated by the Rnorm values. For most situations, enabling all the proposed optimization techniques (*Cfg0*) can realize better timing quality.

1) Timing Quality of Global Placement: When optimization techniques, such as *Cfg1*, *Cfg2*, and *Cfg3*, are disabled, the timing quality can degrade by up to 10%. These results indicate that detailed placement optimization cannot compensate for low-quality global placement. Specifically, disabling blockage-aware spreading and anchor insertion (*Cfg2*) leads to an average increase of 10.9% in CPDs for benchmarks, underscoring the significant impact of placement blockages and the importance of corresponding optimizations.

2) Runtime of Global Placement: Disabling optimization techniques for global placement may reduce runtime by conducting fewer analyses and processes. For instance, disabling blockage-aware spreading and anchor insertion results in an average 10% reduction in runtime for benchmarks since the involved DFS-based clustering is not parallelized. However, despite the resulting runtime overheads, these optimization techniques are justified given the benefits they offer.

3) Timing Quality of Packing and Detailed Placement: Disabling the corresponding techniques, such as *Cfg4*, *Cfg5*, and *Cfg6*, can significantly degrade both the timing quality and runtime of placement. Results obtained with *Cfg4* show that removing the path-length-based factor from Eqn.(13) for global packing increases CPDs by an average of 2.9%. This is because critical paths containing many instances are noticeably disturbed during final packing. Detailed placement using small candidate windows, as implemented in *Cfg5*, results in longer runtimes and fails to reach the optimal critical path delay for most benchmarks. In contrast, sector-guided candidate windows can achieve promising location updates while preserving the proper locations of other instances. On the other hand, using large square candidate windows in *Cfg6* can degrade the timing quality. This is because the serious overlaps of the large windows lead to many candidate site conflicts between instances in one critical path [29].

In Section IV-D, we discussed the importance of temporarily accepting worse CPD during detailed placement to escape from local optima and reach the optimal CPD of final placement. Fig.12 shows two traces of CPD of benchmark FaceDetect [47] during detailed placement. One of them is obtained without allowing CPD degradation during detailed placement, while the other is obtained by temporarily accepting timing downgrading. Although the strict constraint of CPD descending can lead to a smooth CPD optimization trace, tolerating temporary degradation can result in a much better final result.

4) Runtime of Packing and Detailed Placement: Removing the path-length-based factor from Eqn.(13) for global packing, as implemented in *Cfg4*, can significantly increase runtime from a runtime perspective. This is because there are many zigzag critical paths that require a lot of runtime for detailed placement to handle them. Using small square windows without sector windows, as implemented in *Cfg5*, can also

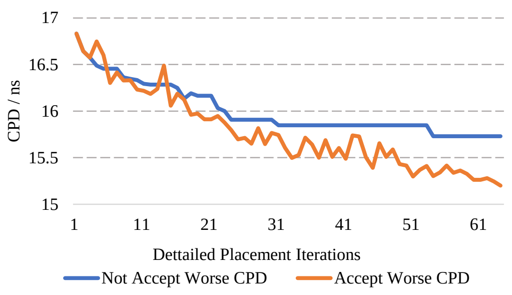


Fig. 12. Example of benchmark FaceDetect [47] shows the benefits of tolerating the temporary degradation of timing during detailed placement.

increase runtime since these narrow windows slow down the optimization progress of critical paths. Meanwhile, the results of timing quality with large square windows for candidate selection (*Cfg6*) can be realized by using sector windows for candidate selection with lower runtime.

D. Portability to Commercial Tools

A set of toolchains is available for interacting with Vivado, such as extracting information on various netlist and device models, which can assist users in handling other designs and devices.

Placement generated by AMF-Placer can be loaded into Vivado via a Tcl script for routing, and all placements generated by the proposed placement flow in this paper can be successfully routed. However, due to license restrictions in the Vivado patch used in the ISPD 2015/2016 contests, we are unable to obtain the routed wirelength from Vivado despite the availability of timing reports.

For more detailed statistics and usage of AMF-Placer 2.0, please refer to the open-source project documentation where we hope that the tool can help people in the community keep up with the progress of commercial tools.

VI. Conclusion

In this paper, we introduce AMF-Placer 2.0, an open-source FPGA placer for complex practical designs with various FPGA resources. AMF-Placer 2.0 utilizes new techniques for timing optimization, including a simple timing model, placement-blockage-aware anchor insertion, WNS-aware timing-driven quadratic placement, and sector-guided detailed placement. Experimental results show that AMF-Placer 2.0 achieves only a 2.3% higher critical path delay (CPD) and 11.5% lower runtime on average than Vivado 2020.2, and a 0.69% higher CPD and 7.0% lower runtime on average than Vivado 2021.2. The source code, Wiki, open-source benchmarks, and checkpoint/log files for the experiments are available at https://github.com/zslwyuan/AMF-Placer.

Acknowledgment

The authors sincerely appreciate the kindly suggestions from reviewers, detailed explanations of UTPlaceF [37] from Dr. Wuxi Li, and useful advice on Vivado metric usages from Dr. Stephen Yang [16], and detailed feedbacks from Mr. Jing Mai, the gold user of AMF-Placer 1.0. Furthermore, the authors acknowledge support from the Hong Kong Research Grants Council General Research Fund (GRF No. 16213521).

References

[1] Xilinx, "Ug974: Ultrascale architecturelibraries guide," 2020.

[2] C. Lavin, M. Padilla, J. Lamprecht, P. Lundrigan, B. Nelson, and B. Hutchings, "Hmflow: Accelerating fpga compilation with hard macros for rapid prototyping," in *2011 IEEE 19th Annual International Symposium on Field-Programmable Custom Computing Machines*. IEEE, 2011, pp. 117–124.

[3] C. M. Fuller, S. W. Gould, S. P. Hartman, E. E. Millham, and G. Yasar, "Field programmable gate arrays using semi-hard multicell macros," Jun. 2 1998, uS Patent 5,761,078.

[4] Xilinx, "Ug904: Vivado design suite user guide: Implementation," 2022.

[5] K. E. Murray, O. Petelin, S. Zhong, J. M. Wang, M. Eldafrawy, J.-P. Legault, E. Sha, A. G. Graham, J. Wu, M. J. Walker *et al.*, "Vtr 8: High-performance cad and customizable fpga architecture modelling," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 13, no. 2, pp. 1–55, 2020.

[6] G. Chen and J. Cong, "Simultaneous placement with clustering and duplication," in *Proceedings of the 41st annual Design Automation Conference*, 2004, pp. 740–772.

[7] M. Gort and J. H. Anderson, "Analytical placement for heterogeneous fpgas," in *22nd international conference on field programmable logic and applications (FPL)*. IEEE, 2012, pp. 143–150.

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

[9] 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*, vol. 37, no. 4, pp. 869–882, 2017.

[10] W. Li, Y. Lin, M. Li, S. Dhar, and D. Z. Pan, "Utplacef 2.0: A high-performance clock-aware fpga placement engine," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 23, no. 4, pp. 1–23, 2018.

[11] C.-W. Pui, G. Chen, W.-K. Chow, K.-C. Lam, J. Kuang, P. Tu, H. Zhang, E. F. Young, and B. Yu, "Ripplefpga: A routability-driven placement for large-scale heterogeneous fpgas," in *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2016, pp. 1–8.

[12] R. Pattison, Z. Abuowaimer, S. Areibi, G. Gréwal, and A. Vannelli, "Gplace: A congestion-aware placement tool for ultrascale fpgas," in *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2016, pp. 1–7.

[13] Y.-C. Kuo, C.-C. Huang, S.-C. Chen, C.-H. Chiang, Y.-W. Chang, and S.-Y. Kuo, "Clock-aware placement for large-scale heterogeneous fpgas," in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2017, pp. 519–526.

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

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

[16] S. Yang, A. Gayasen, C. Mulpuri, S. Reddy, and R. Aggarwal, "Routability-driven fpga placement contest," in *Proceedings of the 2016 on International Symposium on Physical Design*, 2016, pp. 139–143.

[17] S. Yang, C. Mulpuri, S. Reddy, M. Kalase, S. Dasasathyan, M. E. Dehkordi, M. Tom, and R. Aggarwal, "Clock-aware fpga placement contest," in *Proceedings of the 2017 ACM on International Symposium on Physical Design*, 2017, pp. 159–164.

[18] P. Liao, S. Liu, Z. Chen, W. Lv, Y. Lin, and B. Yu, "Dreamplace 4.0: timing-driven global placement with momentum-based net weighting," in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2022, pp. 939–944.

[19] D. Hyun, Y. Fan, and Y. Shin, "Accurate wirelength prediction for placement-aware synthesis through machine learning," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 324–327.

[20] N. Viswanathan, G.-J. Nam, J. A. Roy, Z. Li, C. J. Alpert, S. Ramji, and C. Chu, "Itop: Integrating timing optimization within placement," in *Proceedings of the 19th International Symposium on Physical Design*, ser. ISPD '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 83–90. [Online]. Available: https://doi.org/10.1145/1735023.1735048

[21] A. Bock, S. Held, N. Kämmerling, and U. Schorr, "Local search algorithms for timing-driven placement under arbitrary delay models," in *Proceedings of the 52nd Annual Design Automation Conference*, ser. DAC '15. New York, NY, USA: Association for Computing Machinery, 2015. [Online]. Available: https://doi.org/10.1145/2744769.2744867

[22] V. Livramento, R. Netto, C. Guth, J. L. Güntzel, and L. C. V. D. Santos, "Clock-tree-aware incremental timing-driven placement," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 21, no. 3, apr 2016. [Online]. Available: https://doi.org/10.1145/2858793

[23] G. Wu and C. Chu, "Two approaches for timing-driven placement by lagrangian relaxation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 12, pp. 2093–2105, 2017.

[24] T. Ajayi, V. A. Chhabria, M. Fogaça, S. Hashemi, A. Hosny, A. B. Kahng, M. Kim, J. Lee, U. Mallappa, M. Neseem *et al.*, "Toward an open-source digital flow: First learnings from the openroad project," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–4.

[25] D. Mangiras, A. Stefanidis, I. Seitanidis, C. Nicopoulos, and G. Dimitrakopoulos, "Timing-driven placement optimization facilitated by timing-compatibility flip-flop clustering," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 10, pp. 2835–2848, 2020.

[26] Y.-C. Lu, S. Pentapati, and S. K. Lim, "The law of attraction: Affinity-aware placement optimization using graph neural networks," in *Proceedings of the 2021 International Symposium on Physical Design*, 2021, pp. 7–14.

[27] B. N. Trombley, N. D. Hieter, and D. A. Gay, "Path-based timing driven placement using iterative pseudo netlist changes," Mar. 29 2022, uS Patent 11,288,425.

[28] S.-Y. Chen and Y.-W. Chang, "Routing-architecture-aware analytical placement for heterogeneous fpgas," in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2015, pp. 1–6.

[29] S. Dhar, M. A. Iyer, S. Adya, L. Singhal, N. Rubanov, and D. Z. Pan, "An effective timing-driven detailed placement algorithm for fpgas," in *Proceedings of the 2017 ACM on International Symposium on Physical Design*, 2017, pp. 151–157.

[30] Z. Lin, Y. Xie, G. Qian, S. Wang, J. Yu, and J. Chen, "An analytical timing-driven placer for heterogeneous fpgas: late breaking results," in *Proceedings of the 57th ACM/EDAC/IEEE Design Automation Conference*, 2020, pp. 1–2.

[31] 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 *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2021, pp. 1564–1569.

[32] S. Nikolić, G. Zgheib, and P. Ienne, "Detailed placement for dedicated lut-level fpga interconnect," *ACM Trans. Reconfigurable Technol. Syst.*, nov 2021, just Accepted. [Online]. Available: https://doi.org/10.1145/3501802

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

[34] Xilinx, "Ug574: Ultrascale architecture configurable logic block," 2017.

[35] ——, "Ug573: Ultrascale architecture memory resources," 2021.

[36] ——, "Ug579: Ultrascale architecture dsp slice," 2020.

[37] 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*, vol. 38, no. 11, pp. 2113–2126, 2018.

[38] T.-W. Huang and M. D. Wong, "Opentimer: A high-performance timing analysis tool," in *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2015, pp. 895–902.

[39] D. Stroobandt, J. Depreitere, and J. Van Campenhout, "Generating new benchmark designs using a multi-terminal net model," *Integration*, vol. 27, no. 2, pp. 113–129, 1999.

[40] Ü. V. Çatalyürek and C. Aykanat, "Patoh (partitioning tool for hypergraphs)," in *Encyclopedia of Parallel Computing*. Springer, 2011, pp. 1479–1487.

[41] J. Chen, Z. Lin, Y.-C. Kuo, C.-C. Huang, Y.-W. Chang, S.-C. Chen, C.-H. Chiang, and S.-Y. Kuo, "Clock-aware placement for large-scale heterogeneous fpgas," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 12, pp. 5042–5055, 2020.

[42] P. Maidee, C. Neely, A. Kaviani, and C. Lavin, "An open-source lightweight timing model for rapidwright," in *2019 International Conference on Field-Programmable Technology (ICFPT)*. IEEE, 2019, pp. 171–178.

[43] Y. Zhou, P. Maidee, C. Lavin, A. Kaviani, and D. Stroobandt, "Rwroute: An open-source timing-driven router for commercial fpgas,"

ACM Trans. Reconfigurable Technol. Syst., vol. 15, no. 1, nov 2021. [Online]. Available: https://doi.org/10.1145/3491236

[44] T. Martin, D. Maarouf, Z. Abuowaimer, A. Alhyari, G. Grewal, and S. Areibi, "A flat timing-driven placement flow for modern fpgas," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[45] P. Spindler, U. Schlichtmann, and F. M. Johannes, "Kraftwerk2—a fast force-directed quadratic placement approach using an accurate net model," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 8, pp. 1398–1411, 2008.

[46] G. Guennebaud, B. Jacob *et al.*, "The eigen 3 c++ library," 2010.

[47] Y. Zhou, U. Gupta, S. Dai, R. Zhao, N. Srivastava, H. Jin, J. Featherston, Y.-H. Lai, G. Liu, G. A. Velasquez *et al.*, "Rosetta: A realistic high-level synthesis benchmark suite for software programmable fpgas," in *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2018, pp. 269–278.

[48] Z. Lin, Y. Xie, P. Zou, S. Wang, J. Yu, and J. Chen, "An incremental placement flow for advanced fpgas with timing awareness," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 9, pp. 3092–3103, 2022.

[49] K. Kara, "Spoonn: Fpga-based neural network inference library," 2018.

[50] S. Sukhbaatar, A. Szlam, J. Weston, and R. Fergus, "End-to-end memory networks," *arXiv preprint arXiv:1503.08895*, 2015.

[51] V. Rybalkin, N. Wehn, M. R. Yousefi, and D. Stricker, "Hardware architecture of bidirectional long short-term memory neural network for optical character recognition," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. IEEE, 2017, pp. 1390–1395.

[52] J. Balkind, M. McKeown, Y. Fu, T. Nguyen, Y. Zhou, A. Lavrov, M. Shahrad, A. Fuchs, S. Payne, X. Liang *et al.*, "Openpiton: An open source manycore research framework," *ACM SIGPLAN Notices*, vol. 51, no. 4, pp. 217–232, 2016.

[53] L. Guo, J. Lau, Z. Ruan, P. Wei, and J. Cong, "Hardware acceleration of long read pairwise overlapping in genome sequencing: A race between fpga and gpu," in *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2019, pp. 127–135.

[54] S. Wallentowitz, P. Wagner, M. Tempelmeier, T. Wild, and A. Herkersdorf, "Open tiled manycore system-on-chip," *arXiv preprint arXiv:1304.5081*, 2013.

[55] C. Lavin and A. Kaviani, "Rapidwright: Enabling custom crafted implementations for fpgas," in *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2018, pp. 133–140.

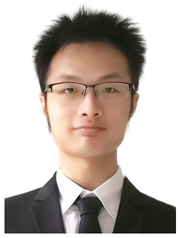
[56] L. Guo, J. Lau, Z. Ruan, P. Wei, and J. Cong, "Hardware acceleration of long read pairwise overlapping in genome sequencing: A race between fpga and gpu," in *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2019, pp. 127–135.

[57] T. Martin, G. Grewal, and S. Areibi, "A machine learning approach to predict timing delays during fpga placement," in *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2021, pp. 124–127.

[58] K. Zhu and D. Wong, "Clock skew minimization during fpga placement," in *Proceedings of the 31st annual Design Automation Conference*, 1994, pp. 232–237.

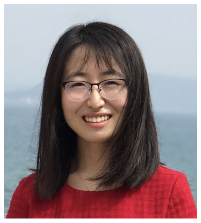
[59] J.-M. Lin, W.-F. Huang, Y.-C. Chen, Y.-T. Wang, and P.-W. Wang, "Dapa: A dataflow-aware analytical placement algorithm for modern mixed-size circuit designs," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2021, pp. 1–8.

[60] D. Fang, B. Zhang, H. Hu, W. Li, B. Yuan, and J. Hu, "Global placement exploiting soft 2d regularity," in *Proceedings of the 2022 International Symposium on Physical Design*, ser. ISPD '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 203–210. [Online]. Available: https://doi.org/10.1145/3505170.3506723



Tingyuan Liang received his B.S. degree in Electrical and Information Engineering from Zhejiang University in 2017. After completing his undergraduate studies, he pursued advanced degrees and earned his MPhil and PhD degrees in the Department of Electronic and Computer Engineering at the Hong Kong University of Science and Technology in 2019 and 2023, respectively. His current research interests include EDA algorithms and computer architecture.

Gengjie Chen obtained his Ph.D. degree, under the supervision of Prof. Evangeline F. Y. Young, from the Department of Computer Science and Engineering of The Chinese University of Hong Kong (CUHK) in 2019. Before that, he received his bachelor degree in the Department of Electronic and Communication Engineering from Sun Yat-sen University (SYSU) in 2015. His research interests include electronic design automation (EDA), combinatorial optimization, numerical optimization, and machine learning.



Jieru Zhao (M'22) received the Ph.D. degree in electronic and computer engineering from the Hong Kong University of Science and Technology, Hong Kong, in 2020. She is an Assistant Professor with the Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China. Her current research interests include reconfigurable system, high-level synthesis and hardware-software co-design for emerging applications. Dr. Zhao was a recipient of the Best Paper Award at ICCAD 2017 and Best Paper Nominations at DATE 2022, SC 2021, and CASES 2018.

Sharad Sinha (S'03, M'14) is an associate professor with Dept. of Computer Science and Engineering, Indian Institute of Technology (IIT) Goa. Previously, he was a Research Scientist at NTU, Singapore. He received his PhD degree in Computer Engineering from NTU, Singapore (2014). He received the Best Speaker Award from IEEE CASS Society, Singapore Chapter, in 2013 for his PhD work on High Level Synthesis and serves as an Associate Editor for IEEE Potentials and ACM Ubiquity. Dr. Sinha earned a Bachelor of Technology (B.Tech) degree in Electronics and Communication Engineering from Cochin University of Science and Technology (CUSAT), India in 2007. From 2007-2009, he was a design engineer with Processor Systems (India) Pvt. Ltd. Dr. Sinha's research and teaching interests are in computer architecture, embedded systems and reconfigurable computing.

Wei Zhang Wei Zhang (M'05) received a Ph.D. degree from Princeton University, Princeton, NJ, USA, in 2009. She was an assistant professor with the School of Computer Engineering, Nanyang Technological University, Singapore, from 2010 to 2013. Dr. Zhang joined the Hong Kong University of Science and Technology, Hong Kong, in 2013, where she is currently a professor and she established the reconfigurable computing system laboratory (RCSL). Dr. Zhang has authored or co-authored over 80 book chapters and papers in peer reviewed journals and international conferences. Dr. Zhang serves as the Associate Editor for TECS, TVLSI, and JETC. She also serves on many organization committees and technical program committees. Dr. Zhang's current research interests include reconfigurable sys- tems, FPGA-based design, low-power high-performance multicore systems, electronic design automation, embedded systems, and emerging technologies.