UTPlaceF: A Routability-Driven FPGA Placer with Physical and Congestion Aware Packing

Invited Paper

Wuxi Li , Shounak Dhar , and David Z. Pan

ECE Department, University of Texas at Austin, Austin, TX, USA
{wuxi.li, shounak.dhar}@utexas.edu; dpan@ece.utexas.edu

ABSTRACT

FPGA packing and placement without routability consideration could lead to unroutable results for high-utilization designs. Conventional FPGA packing and placement approaches are shown to have severe difficulties to yield good routability. In this paper, we propose a FPGA packing and placement engine called UTPlaceF that simultaneously optimizes wirelength and routability. A novel physical and congestion aware packing algorithm and several congestion aware detailed placement techniques are proposed. Compared with the top 3 winners of ISPD'16 FPGA placement contest, UTPlaceF can achieve 3.3%, 7.7% and 28.3% better routed wirelength with similar or shorter runtime.

1. INTRODUCTION

The *field programmable gate array (FPGA)* is a type of premanufactured integrated circuit designed to be configured by customers or designers. FPGAs are becoming more and more popular nowadays because of their ability to re-program in the field to fix bugs, shorter time to market, and lower non-recurring engineering costs. Historically, FPGAs was only used for fast realization of small digital circuits. However, in recent years, the gate count of commercial FPGAs has reached scale of millions [1], so much more complex digital systems are moving towards FPGA-based design methodologies.

A representative FPGA CAD flow is shown in Fig. 1. During logic synthesis and techcnology mapping, a circuit is translated into a netlist composed of *lookup tables (LUTs)* and *flip-flops (FFs)*. In the packing stage, several LUTs and FFs together form a *basic logic element (BLE)* and then several BLEs are grouped into a *configurable logic block (CLB)*. After packing, placement is responsible for determining the physical position of all CLBs in a two-dimensional array while optimizing some metrics (e.g. wirelength, routability, timing, power, and etc.). Finally, routing is performed to physically connect CLBs.

As design size and complexity continue to increase dramatically, routability has become an important metric in FPGA domain. Traditional pure wirelength-driven optimizations without routability consideration often failed to map circuits into FPGA devices. Among all CAD stages, packing and placement play key roles in optimizing various metrics, particularly routability.

Packing algorithms typically can be divided into three different categories: 1) seed-based approaches, 2) partitioning-based approaches, and 3) placement-guided and cluster-merging-based approaches. Seed-based packing approaches iteratively choose a BLE to form an initial CLB, then keep adding other unpacked BLEs into the CLB based on an attraction function until no more BLEs can be added. VPack [2], T-VPack [3], RPack [4], iRAC [5], T-NDPack [6], and MO-Pack [7] are representative examples of seed-based algorithms with different objectives and attraction functions. Partitioning-based approaches, like [8] and PPack [9], first apply a k-way partitioning to get a set of potential CLBs, and then perform a sequence of inter-partition moves to legalize the packing solution. HDPack [10] is an example of placement-guided and cluster-merging-based methods. It incorporates physical information using a min-cut-partitioning based global placement, and applies the idea of *hybrid first choice clustering (HFCC)* from [11] to recursively group clusters with highest attraction until no more merging could be performed.

To improve routability, packing algorithms like [12], iRAC, [13], Un/DoPack [14], and T-NDPack proposed several different depopulation techniques to prevent CLBs from being fully filled. Depopulation can be classified into two categories, uniform depopulation and non-uniform depopulation. iRAC is a good example of uniform depopulation, it limits cell utilization of all CLBs based on Rent's rule. Un/DoPack is a example of non-uniform depopulation. It first runs through a regular CAD flow, then depopulates CLBs in the congested regions based on the routing result.

FPGA placement algorithms are very similar to ASIC's placement and typically fall into one of the following three categories: 1) simulated-annealing-based approaches, 2) min-cut-partitioning-based approaches, and 3) analytical approaches. Simulated annealing based placers, like VPR [15], apply a probabilistic searching to approximate the global optimal solution. Min-cut-partitioning based placers, e.g. [16,17], recursively apply min-cut partitioning until cells are fully spread out. Analytical placers typically ap-

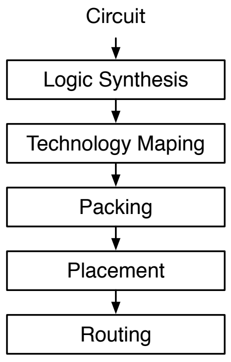


Figure 1: A representative FPGA CAD flow.

ICCAD '16, November 07-10, 2016, Austin, TX, USA
© 2016 ACM. ISBN 978-1-4503-4466-1/16/11...$15.00
DOI: http://dx.doi.org/10.1145/2966986.2980083

proximate cost metrics, like wirelength and bin density, with a smooth objective function, then use numeric solvers to find the optimal solution. Different analytical placement approaches have been extensively studied in [18–23].

1.1 Motivation

Packing and placement are two key steps to achieve high-quality physical implementation with good routability. However, we found that existing academic packing and placement approaches have the following limitations:

- Existing packing algorithms do not have good knowledge of cells' physical locations. Logical packing, which performs packing based only on logical connectivity, could cluster cells that are physically far apart. As a result, it may lead to wirelength-unfriendly netlists and worsen routability. Most seed-based and partitioning-based packers do not have physical location information and only perform logical clustering. Existing placement-guided packers only have rough global placements, which is of poor quality compared to state-of-art placement engines. We believe that accurate physical information could better guide packing and yield placement-friendly CLB level netlists.
- Existing packing algorithms are unaware of actual congestion information, which is crucial for efficient and high-quality depopulation. Blindly applying uniform depopulation would inevitably worsen wirelength and area, and it is more efficient to only avoid overpacking in routing congested regions. Therefore accurate routing congestion information is of great importance in packing stage.
- In recent years, the gate count in modern FPGAs already reached scale of millions. Therefore, packing and placement algorithms with high-quality and good scalability are highly desirable for large scale FPGAs.

1.2 Contributions

In this paper, we propose a new routability-driven FPGA packing and placement engine called UTPlaceF. Our main contributions are listed as follows:

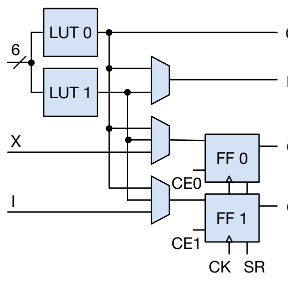
- We propose a novel packing algorithm that incorporates accurate physical information based on a high-quality analytical global placement.
- We propose a routing congestion aware depopulation technique to efficiently balance wirelength and routability in a correct by construction manner.
- We propose several congestion aware detailed placement techniques to improve wirelength without degrading routability.
- We perform experiments on the *ISPD'16 Routability-Driven FPGA Placement Contest* [24] benchmark suite released by Xilinx. Compared to the ISPD'16 contest top 3 winners, UTPlaceF achieves 3.3%, 7.7%, and 28.3% better routed wirelength with similar or shorter runtime.

The rest of this paper is organized as follows: Section 2 reviews the preliminaries and presents the UTPlaceF framework overview. Section 3 and Section 4 give the details of UTPlaceF packing and placement algorithms, respectively. Section 5 shows the experimental results, followed by the conclusion in Section 6.

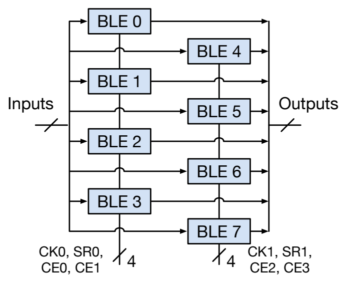
2. PRELIMINARIES AND OVERVIEW

2.1 FPGA Architecture

In ISPD'16 FPGA placement contest, all benchmarks are targeted to Xilinx Ultrascale VU095. The architecture of BLE and CLB in this FPGA are shown in Fig. 2. Each CLB has eight BLE sites, and each BLE contains two LUT sites and two FF sites.



(a) BLE



(b) CLB

Figure 2: BLE and CLB in Xilinx Ultrascale architecture.

The two LUT sites in a BLE could be implemented as a single 6-input LUT or two smaller LUTs with total number of different input pins less than 6. The two FFs in a BLE must share the same clock and set/reset pins, however their clock enable pins could be different. There are two clock, two set/reset, and four clock enable pins available for each CLB. Each clock, set/reset and each two clock enables are dedicated to 4 BLEs. More details can be found in [24].

2.2 Quadratic Placement

A FPGA netlist can be represented as a hypergraph $H = (V, E)$, where $V = \{v_1, v_2, ..., v_{|V|}\}$ is the set of cells, and $E = \{e_1, e_2, ..., e_{|E|}\}$ is the set of nets. Let $\boldsymbol{x} = \{x_1, x_2, ..., x_{|V|}\}$ and $\boldsymbol{y} = \{y_1, y_2, ..., y_{|V|}\}$ be the x and y coordinates of all cells. The wirelength-driven global placement problem is to determine position vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ that minimize the total wirelength and obey bin density constraint. Wirelength is measured by the HPWL,

$$HPWL(\boldsymbol{x}, \boldsymbol{y}) = \sum_{e \in E} \{\max_{i,j \in e} |x_i - x_j| + \max_{i,j \in e} |y_i - y_j|\} \quad (1)$$

As HPWL is not differentiable everywhere, quadratic placers approximate it by squared Euclidean distance between cells. Therefore, the wirelength cost function in quadratic placer is defined as,

$$W(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{2}\boldsymbol{x}^T Q_x \boldsymbol{x} + \boldsymbol{c}_x^T \boldsymbol{x} + \frac{1}{2}\boldsymbol{y}^T Q_y \boldsymbol{y} + \boldsymbol{c}_y^T \boldsymbol{y} + const \quad (2)$$

2.3 UTPlaceF Overview

Fig. 3 shows the flowchart of UTPlaceF. The overall flow is composed of three parts: 1) *physical and congestion aware packing (PCAP)*, 2) global placement, and 3) detailed placement.

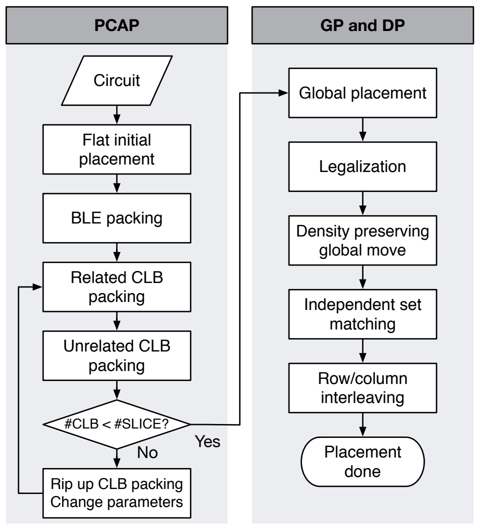


Figure 3: Overview of UTPlaceF.

In PCAP, a *flat initial placement (FIP)* is generated to better guide packing. FIP is responsible for generating cells' physical locations, and detecting cells that are likely to be placed into routing congested regions. In the packing stage of PCAP, first LUTs and FFs are grouped into BLEs, then BLEs are clustered into CLBs. We assume that FIP yields the optimal cell relative position, and packing should not perturb it too much. Therefore PCAP, with cell physical locations information, disallows long-distance packing and prefers close packing. Similar to iRAC, absorbing small nets is treated as the main objective during packing stage of PCAP to reduce channel width and routing demand, which in turn improves wirelength and routability. Besides considering grouping connected cells, UTPlaceF also packs unconnected cells based on their physical locations to further reduce number of CLBs. Leveraged by routing congestion information from FIP, PCAP can perform loose packing only for cells that are likely to be placed into routing hotspots, and avoid blindly depopulating throughout whole netlists for routability enhancement. By using this congestion aware depopulation technique, PCAP is able to achieve both good wirelength and routability.

Our global placement basically shares the same framework with FIP but handles CLBs instead of LUTs and FFs. In detailed placement stage, a bipartite-matching-based minimum-movement legalization is applied first. Then density preserving global move, independent set matching, and row/column interleaving are consecutively performed to further reduce wirelength. To preserve the routability optimized global placement solution, white spaces in congested regions are handled specially throughout the detailed placement stage.

3. PHYSICAL AND CONGESTION AWARE PACKING

3.1 Flat Initial Placement

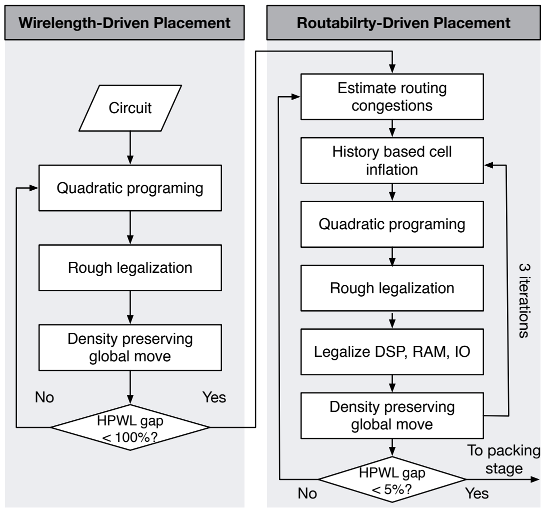


Figure 4: Overall flow of the flat initial placement.

Our FIP adopts the main framework of an ASIC placer *POLAR 2.0* [25]. Its overall flow is shown in Fig. 4. In each iteration of wirelength-driven placement, a quadratic program is solved followed by rough legalization [26] for reducing cells overlaps. Then the density preserving global move [27] is applied to improve the wirelength of the rough legalized placement while preserving bin densities. A sequence of pure wirelength-driven placement iterations is performed until the gap between the upper bound wirelength and the lower bound wirelength is less than a certain number, which is 100% in PCAP. In the routability-driven placement stage, after a certain number of placement iterations, a fast global router NCTUgr [28] is called for routing congestion estimation, then similar to POLAR 2.0, cells in congested regions are inflated by a small ratio, and the inflation accumulates to the end of FIP.

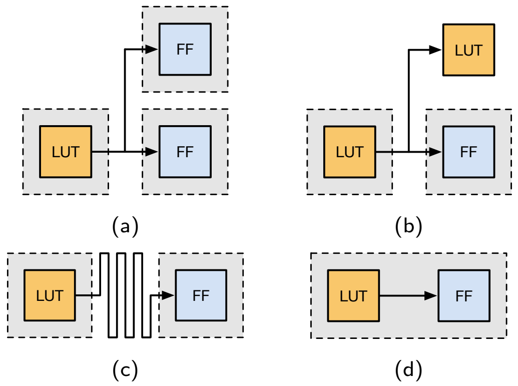


Figure 5: Different LUT and FF paring scenarios. (a) and (b): LUT fanouts to multiple celles. (c) LUT fanouts to only one far FF. (d) LUT fanouts to only one close FF.

The two objectives of FIP are 1) generating physical location for each LUT and FF, and 2) detecting LUTs and FFs that are in routing congested regions. Cell physical locations are explicitly generated by the wirelength and routability co-optimized placement. Congestion information associated with cells is implicitly obtained from history based cell inflation. The insight of cell inflation is quantifying the possibility of a cell lying in routing congested regions using its area – larger cell area indicates larger possibility of being placed into congested regions. After FIP, each LUT and FF would have a physical location and a cell area, which indicates the congestion level associated with it.

Besides LUTs and FFs, FPGAs also contain cells, like DSPs, RAMs, and IOs, that have their specific physical sites. Typically, sites for these cells are discrete and scattered on FPGAs. If we handle them like LUTs and FFs in rough legalization, they might be far away from their legal positions in the final FIP solution. This discrepancy would introduce inaccuracy of cell relative position into FIP. To eliminate this discrepancy, UTPlaceF performs an extra legalization step for DSPs, RAMs, and IOs right after the conventional rough legalization in the second stage of FIP. By simply doing this, DSPs, RAMs, and IOs will use their legal positions as their anchor points in placement iterations, and the discrepancy will be eliminated in the final FIP solution. The legalization approach here is similar to our CLB legalization discussed in Section 4.2.

3.2 Max-Weighted-Matching-Based BLE Packing

As a BLE in our target FPGA architecture, Xilinx Ultrascale, contains 2 LUTs and 2 FFs, existing VPR-style BLE packing cannot be directly applied. To address this new BLE architecture, we propose a two step BLE packing algorithm that comprises: 1) LUT and FF pairing, and 2) LUT, FF pairs matching.

In PCAP, we apply the LUT and FF paring in a similar manner to the BLE packing in VPack. As shown in Fig.5, we group a LUT and a FF together if the LUT only fanouts to the FF. Besides that, long-distance packing is rejected, and only packing within *maximum packing distance of BLE* ($\overline{\lambda_b}$) is allowed.

In the second step, *max-weighted matching* is used for finalizing the BLE packing. We construct an undirected weighted graph $UWG = (V, E)$, where each v_i in $V = v_1, v_2, ..., v_{|V|}$ is a LUT/FF pair, a single LUT, or a single FF. $E = e_1, e_2, ..., e_{|E|}$ represents the set of legal mergings. To apply high-attraction and close packing, we say a merging (v_i, v_j) is legal if and only if,

1. v_i and v_j are connected in the netlist.
2. Merging v_i and v_j into the same BLE does not violate any packing rules.
3. The merging attraction is greater than the *minimum packing attraction of BLE* ($\underline{\phi_b}$).

In the UWG, edge weights are set as merging attractions. The attraction value for any merging is defined as,

$$\phi_b(v_i, v_j) = (1 - e^{\gamma_b(dist(v_i,v_j)-\overline{\lambda_b})}) \sum_{p \in Net(v_i \cap v_j)} \frac{k_b}{deg(p)-1} \quad (3)$$

where γ_b is a constant value being experimentally set as 0.2, $dist(v_i, v_j)$ is the Manhattan distance between v_i and v_j, $\overline{\lambda_b}$ is the maximum packing distance of BLE which is 4 in PCAP, $Net(c_i \cap c_j)$ is the set of nets shared between v_i and v_j, $deg(p)$ is the total number of pins of net p that is exposed in cluster level, and k_b is 2 for 2-pin nets and 1 for other nets.

The first term, $1 - e^{\gamma_b(dist(v_i,v_j)-\overline{\lambda_b})}$, is a packing distance penalty factor in range $(-\infty, 1)$. This factor is very close to 1 for mergings of distance much less than $\overline{\lambda_b}$. It drops quickly as $dist(v_i, v_j)$ gets close to $\overline{\lambda_b}$, and becomes negative once $dist(v_i, v_j)$ is greater than $\overline{\lambda_b}$. By using the first term, short-distance mergings in PCAP are always preferred. The second term, $\sum_{p \in Net(v_i \cap v_j)} \frac{k_b}{degree_p - 1}$, is introduced for reducing number of nets exposed in cluster level, which in turn improves wirelength and routability. With the second term, merging two clusters that share more small nets is of high priority, and 2-pin nets are given even higher weight by the factor $k_b = 2$.

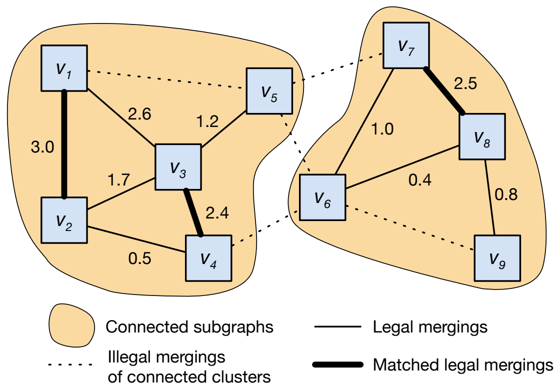


Figure 6: A simple max-weighted cluster matching example.

Fig. 6 shows a simple cluster matching example. Due to our rules for legal merging, the constructed UWG typically is not connected and comprises many small connected subgraphs. Mergings in different connected subgraphs are independent, so PCAP performs a max-weighted matching algorithm on each of these subgraphs and all matched cluster pairs would be merged. The location of a merged cluster is set as the average location of all cells (LUTs and FFs) it contains, and the cluster area is simply the sum of cell areas. After each pass of matching and merging, PCAP rebuilds the UWG and re-solves the max-weighted matching for each new connected subgraph until no more legal merging exists. The pseudo-code of our BLE packing is summarized in Alg. 1.

3.3 Related CLB Packing with Congestion Aware Depopulation

After BLE packing, we create a CLB for each single BLE. CLB packing is done by successively merging smaller CLBs into larger ones. CLBs that share common nets are said to be related, and in this stage only related CLBs mergings are considered.

The *BestChoice Clustering (BC)* [29] is used as our main engine for related CLB packing. In BC, the attractions of all legal CLB mergings are calculated first, then the algorithm iteratively merges CLB pairs with the highest attraction. The location and area of a merged CLB is set as the average location and total area of cells it contains, respectively. After each merging, the legality and attraction of mergings related to the new CLB are updated accordingly. The pseudo-code of our BC-based related CLB packing is summarized in Alg. 2.

Algorithm 1 Max-Weighted-Matching-Based BLE Packing

```
Input: Post-FIP netlist.
Output: BLE level netlist with external nets reduced.
 1: Pair LUTs and FFs;
 2: while legal cluster merging exists do
 3:     Construct an UWG using Eqn. (3);
 4:     for each connected subgraph do
 5:         Run max-weighted matching;
 6:         for each matched edge do
 7:             Merge corresponding clusters;
 8:             Set location and area of the merged cluster;
 9:         end for
10:     end for
11: end while
```

In PCAP, a related CLB merging (c_i, c_j) is said to be legal if and only if

1. c_i and c_j are connected in the netlist.
2. Merging c_i and c_j into the same CLB does not violate any packing rules.
3. The merging attraction is greater than *the minimum packing attraction of related CLB* ($\underline{\phi_{rc}}$).
4. Total area of c_i and c_j is no grater than *maximum CLB area* ($\overline{a_c}$).

The first three rules are inherited from our BLE packing. The fourth rule is introduced to perform congestion aware depopulation and avoid overpacking in routing congested regions. As discussed in Section 3.1, cells with larger area have higher possibility to be placed into routing congested regions. By constraining area of each CLB, PCAP would apply loose packing in routing congested regions, and tight packing in other regions. This congestion awareness makes PCAP able to achieve a good trade-off between wirelength and routability. Fig. 7 illustrates the congestion aware depopulation technique in PCAP, note that BLEs with larger area are in routing congested regions.

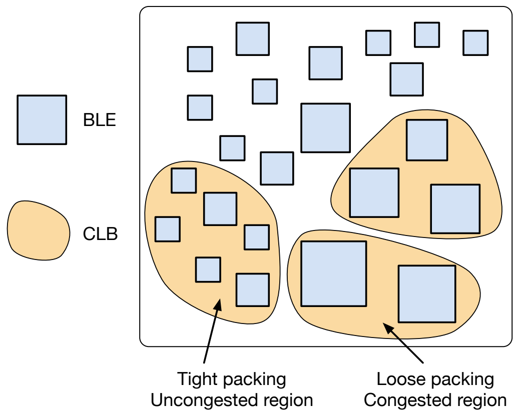


Figure 7: Congestion aware depopulation of PCAP during packing.

The attraction function of related CLB mergings (c_i, c_j) is defined as,

$$\phi_{rc}(c_i, c_j) = (1 - e^{\gamma_{rc}(dist(c_i,c_j)-\overline{\lambda_{rc}})}) \sum_{p \in Net(c_i \cap c_j)} \frac{k_{rc}}{deg(p)-1} \quad (4)$$

Eqn. (4) is basically a replica of our BLE packing attraction function defined in Eqn. (3), but differs only by some constant parameters. We experientially set γ_{rc} to 0.2, and $\overline{\lambda_{rc}}$ to 8. k_{rc} is 2 for 2-pin nets and 1 for other nets.

Algorithm 2 BC-based Related CLB Packing

```
Input: Post-BLE-packing netlist.
Output: CLB level netlist with external nets reduced.
 1: Create an empty priority queue PQ;
 2: Find all legal mergings $(c_i, c_j)$ and their attraction $\phi_{rc}(c_i, c_j)$;
 3: Insert all legal mergings with $\phi_{rc}(c_k, c_j) \geq \underline{\phi_{rc}}$ into PQ;
 4: while PQ is not empty do
 5:     Pick merging $(c_i, c_j)$ with the highest attraction from PQ;
 6:     if $c_i$ or $c_j$ has been merged already then
 7:         continue;
 8:     end if
 9:     Merge $c_i$ and $c_j$ to form $c_{ij}$;
10:     Set location and area of $c_{ij}$;
11:     for each CLB $c_k$ that connects to $c_{ij}$ do
12:         if merging $(c_k, c_{ij})$ violates packing rules then
13:             continue;
14:         end if
15:         Calculate attraction $\phi_{rc}(c_k, c_{ij})$;
16:         if $\phi_{rc}(c_k, c_{ij}) \geq \underline{\phi_{rc}}$ then
17:             Insert $(c_k, c_{ij})$ into PQ;
18:         end if
19:     end for
20: end while
```

3.4 Unrelated CLB Packing

CLBs without common nets are said to be unrelated. After related CLB packing stage, unrealted CLB mergings are considered. Different from related CLB packing, in which reducing external nets is the main objective, unrelated CLB packing aims to reduce number of CLBs.

BC-based approaches typically could yield very good packing solutions for given attraction functions. However, it has an inherent drawback – incapability of making tight packing. Generally, BC would generate a large number of medium-sized clusters that are difficult to merge further due to the cluster capacity constraint. In contrast, seed-based approaches are effective to achieve tight packing but with degradation of packing quality. In PCAP, we apply both strategies properly in different scenarios.

Initially, PCAP performs BC-based unrelated CLB packing in a manner similar to our related CLB packing, but with the following three differences: 1) merging unrelated CLBs could be legal, 2) in Alg. 2 line 2 and line 11, instead of only evaluating related CLB mergings, all mergings within distance $\overline{\lambda_{uc}}$ would be considered, and 3) a different attraction function defined in Eqn. (5) is used.

$$\phi_{uc}(c_i, c_j) = \\ (1 - e^{\gamma_{uc}(dist(c_i,c_j) - \overline{\lambda_{uc}})}) \cdot (N_{BLE}(c_i) + N_{BLE}(c_j)) \quad (5)$$

The attraction function ϕ_{uc} comprises two terms. The first term, $1 - e^{\gamma_{uc}(dist(c_i,c_j) - \overline{\lambda_{uc}})}$, is a packing distance penalty factor similar to Eqn. (3) and Eqn. (4). The second term, $N_{BLE}(c_i) + N_{BLE}(c_j)$, is the total number of BLEs of the two merging CLBs. By applying this term, which gives higher priority to mergings that yield larger CLBs, PCAP can achieve tighter packing even using BC.

For high-utilization designs, the BC-based unrelated CLB packing might still not be able to generate tight enough packing solutions that satisfy FPGA capacity constraint. In this case, existing packing solution will be ripped up, and a seed-based packing will be applied instead of the BC-based approach to aggressively reduce number of CLBs. The details of this rip-up and re-packing phase will be further discussed in Section 3.5.

Note that, although the objective of our unrelated CLB packing is to reduce number of CLBs and yield tight packing, the congestion aware depopulation technique described in Section 3.3 is still applied in this stage to maintained good routability.

3.5 Net Reduction and Packing Tightness Trade-off

Our related CLB packing works effectively to reduce number of external nets, however it often yields relatively loose packing due the the inherent shortcoming of BC mentioned in section 3.4. In contrast, our unrelated CLB packing is capable of aggressively reducing number of CLBs and achieving tight packing. Therefore, if more packing is performed in the related CLB packing stage, a loose packing solution with less external nets would be yielded. However, if we only do a small portion of packing in the related CLB packing stage and leave most of work to unrelated CLB packing, the final packing would be more inclined to the "tight" side with more external nets.

In PCAP, $\underline{\phi_{rc}}$ (the minimum packing attraction of related CLB packing) and $\overline{\lambda_{uc}}$ (the maximum packing distance of unrelated CLB) are used to control the amount of packing work for each (related/unrelated) CLB packing stage. Initially, $\underline{\phi_{rc}}$ is set as 0 to aggressively reduce number of external nets, and $\overline{\lambda_{uc}}$ is set as 8 to only allow close packing in unrelated CLB packing stage. This initial setting typically results in a loose packing with large amount of net reduction. For high utilization designs, however, the packing solution generated by the initial setting could be sparse to the extent that number of CLBs exceeds the FPGA capacity. To address this problem, PCAP would discard the existing CLB packing solution and perform a re-packing step, which applies related and unrelated CLB packing again. In the re-packing phase, however, $\underline{\phi_{rc}}$ is increased to reduce related CLB packing, and $\overline{\lambda_{uc}}$ is also increased to allow unrelated CLB packing of longer distance. As results, the re-packing step would yield tighter packing but sacrifice net reduction. The re-packing step would be repeated until the number of CLBs is less than the FPGA capacity.

For high-utilization designs, barely relying on BC-based unrelated CLB packing cannot guarantee tight enough packing solutions that satisfy FPGA capacity constraint. It would reach saturation points where number of CLBs cannot be further reduced even with larger $\overline{\lambda_{uc}}$. To address this issue, as mentioned in Section 3.4, PCAP will switch to seed-based unrelated CLB packing once $\overline{\lambda_{uc}}$ is larger than a certain value.

4. POST-PACKING PLACEMENT

4.1 Global Placement

After PCAP, the global placement is performed immediately to further optimize wirelength and routability. Our global placement shares the same framework with FIP, but instead of optimizing flat LUT/FF netlist, it considers each CLB as a whole. FIP is used as the starting point for global placement and the routability-driven placement stage is applied directly. To avoid global placement being stuck in the local optimal around FIP, the weight of pseudo-nets for cell spreading is reduced at the beginning of global placement.

4.2 Min-Cost Bipartite Matching Based Legalization

A notable difference between ASIC and FPGA legalization is that ASIC standard cells have different dimensions whereas FPGA CLBs have the same size. Because of this special property, FPGA legalization problem could be formulated as a min-cost-max-cardinality bipartite matching problem with Manhattan distance between CLBs and slices as cost. By solving the corresponding bipartite matching problem, global placement can be legalized with minimum total movement by the nature of this problem. However, solving a complete bipartite matching for large designs is impractical in terms of runtime. To address this problem, we apply min-cost matching in a window-by-window manner so that only a small matching within each window is solved. Windows in congested regions are legalized first, and neighboring available slices would be considered if number of

CLBs is larger than number of available slices within a window. Our legalization approach is summerized in Alg. 3.

Algorithm 3 Min-Cost Bipartite Matching Based Legalization

```
Input: Rough-legalized CLB level netlist.
Output: Legalized placement with minimum movement.
1: Split the placement region into non-overlapping windows;
2: Sort windows by their routing congestion in descending order;
3: for each unlegalzied window do
4:     while Num. CLBs > Num. slices do
5:         Add neighboring unoccupied slices into window;
6:     end while
7:     Run min-cost bipartite matching between CLBs and slices with
       Manhattan distance as cost.
8:     Move each CLB to its matched slice, and set the slice as occu-
       pied.
9: end for
```

4.3 Congestion Aware Independent Set Matching

The idea of bipartite matching can also be applied to optimize wirelength. For a given set of legalized CLBs, a wirelength optimization problem could be formulated as a max-weighted bipartite matching with edge weights as HPWL improvement of moving CLBs to different slices. However, solving this matching problem cannot guarantee the optimal HPWL improvement, since the edge weight of a CLB depends on the positions of other connected CLBs in the same matching set. To overcome this drawback, we adopt the *independent set matching (ISM)* idea from NTUPlace3 [30], and only apply matching within a set of CLBs that do not share any common nets. Based on that, spaces are also considered in our matching to further increase the solution space.

The ISM works effectively for optimizing HPWL. However it could ruin the local CLB density optimized for routability, especially when spaces are considered in our ISM. To mitigate this problem, we propose a congestion aware ISM with three extra constraints introduced: 1) CLBs can be moved out of but not into routing congested regions, 2) spaces can be moved into but not out of congested regions, and 3) moves within congested regions are disallowed. Fig. 8 shows a simple matching example with the extra constraints applied. To get accurate congestion information, the routing congestion map is updated after a certain number of ISM iterations. By applying our congestion aware ISM, HPWL could be optimized without routability degradation.

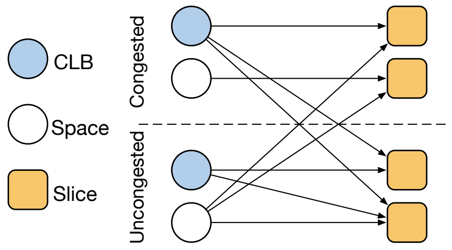


Figure 8: Illustration of our congestion aware ISM.

4.4 Other Detailed Placement Techniques

Besides ISM, UTPlaceF also applies two other effective detailed placement techniques, global move and cell interleaving [31]. We borrow the density preserving global move idea from [27] and extend it to detailed placement by only moving cells to legal positions in their optimal regions. Cell interleaving is performed in both rows and columns because of the regularity of FPGA slices. To avoid routability being degraded, spaces in routing congested regions are fixed during cell interleaving.

5. EXPERIMENTAL RESULTS

Table 1: ISPD'16 Placement Contest Benchmarks Statistics

Benchmark	#LUT	#FF	#RAM	#DSP	#Ctrl Set
FPGA-1	50K	55K	0	0	12
FPGA-2	100K	66K	100	100	121
FPGA-3	250K	170K	600	500	1281
FPGA-4	250K	172K	600	500	1281
FPGA-5	250K	174K	600	500	1281
FPGA-6	350K	352K	1000	600	2541
FPGA-7	350K	355K	1000	600	2541
FPGA-8	500K	216K	600	500	1281
FPGA-9	500K	366K	1000	600	2541
FPGA-10	350K	600K	1000	600	2541
FPGA-11	480K	363K	1000	400	2091
FPGA-12	500K	602K	600	500	1281
Resources	538K	1075K	1728	768	N/A

UTPlaceF was implemented in C++ and tested on a Linux machine with 3.40 GHz CPU and 32GB RAM. The benchmark suite released by Xilinix for ISPD'16 FPGA placement contest was used to evaluate the efficiency of UTPlaceF.

5.1 Benchmark Characteristics

The characteristics of ISPD'16 benchmark suite are listed in Table 1. This benchmark suite has numbers of cells ranging from 0.1 to 1.1 million, which are much larger than that of existing academic FPGA benchmarks. Note that several benchmarks have extremely high cell utilization, which raises two requirements to FPGA placement packing and placement engines: 1) the capability to yield tight packing solutions to satisfy the CLB capacity constraint, and 2) the capability to reduce routing resource demand, since little white space is available for cell and routing demand spreading.

5.2 Comparison with Contest Winners

We compare our results with the top 3 winners of ISPD'16 placement contest, and results are shown in Table 2. All routed wirelength are reported by Xilinx Vivado v2015.4, and runtime of the contest winners are evaluated on a Linux Machine with 3.20 GHz CPU and 32GB RAM. Ratios in the last row of Table 2 are based on comparisons with our results, and only benchmarks that the contest winner completed are considered in each comparison. It can be seen that UTPlaceF yields the best overall routed wirelength. On average UTPlaceF outperforms by 3.3%, 7.7%, and 28.3% on routed wirelength compared with the top 3 contest winners, respectively, and note that only UTPlaceF is able to route all 12 benchmarks. In terms of runtime, as UTPlaceF is evaluated in a different machine, it is not fair to compare them directly. However we still can see that the runtime of UTPlaceF is about the same as the first place team, which has the fastest runtime among the top 3 contest winners.

5.3 Runtime Analysis

The runtime breakdown of UTPlaceF is shown in Table 3. On average, about 75.3% of total runtime is token by PCAP, while CLB global placement and detailed placement respectively take about 7.2% and 17.4% of total runtime, and legalization only takes about 0.1% of total runtime. In PCAP, FIP takes about 72.6% of total runtime, and the rest of packing stages take the remaining 2.7%. In detailed placement stage, ISM takes about 13.6% of total runtime, global move takes 0.3%, and cell interleaving takes 3.5% of total runtime.

6. CONCLUSION

With the utilization of FPGA designs being pushed to the upper limit, routability optimization is becoming a fundamental issue in modern physical design flow for FPGA. In this paper we have proposed a routability-driven FPGA packing and placement engine called UTPlaceF. A novel packing algorithm PCAP and several detailed placement techniques for wirelength and routability co-optimization are proposed. The experimental

Table 2: Placement Quality Comparison with ISPD'16 Contest Winners

Benchmark	UTPlaceF		1st Place		2nd Place		3rd Place	
	Routed WL	Runtime(s)	Routed WL	Runtime(s)	Routed WL	Runtime(s)	Routed WL	Runtime(s)
FPGA-1	384709	215	PE*	N/A	379932	118	581975	97
FPGA-2	652690	399	677877	435	679878	208	1046859	191
FPGA-3	3181331	1555	3223042	1527	3660659	1159	5029157	862
FPGA-4	5504083	1289	5628519	1257	6497023	1149	7247233	889
FPGA-5	10068879	1237	10264769	1266	UR	N/A	UR	N/A
FPGA-6	6411247	2827	6630179	2920	7008525	4166	6822707	8613
FPGA-7	10040562	2588	10236827	2703	10415871	4572	10973376	9169
FPGA-8	8113483	2705	8384338	2645	8986361	2942	12299898	2741
FPGA-9	13616625	3407	UR†	N/A	13908997	5833	UR	N/A
FPGA-10	8866049	4091	PE	N/A	PE	N/A	UR	N/A
FPGA-11	10834629	3267	11091383	3227	11713479	7331	UR	N/A
FPGA-12	8246410	4625	9021769	4539	PE	N/A	UR	N/A
Ratio	1.00	1.00	1.033	1.004	1.077	1.505	1.283	1.949

* PE: Placement error

† UR: Unroutable placement

Table 3: Runtime Breakdown of UTPlaceF

Benchmark	PCAP				GP	Legalization	DP			Total
	FIP	BLE Packing	Related CLB Packing	Unrelated CLB Packing			Global Move	ISM	Cell Interleaving	
FPGA-1	130	2	1	1	17	1	1	47	15	215
FPGA-2	265	3	3	1	26	1	1	74	25	399
FPGA-3	1108	7	7	3	106	1	4	228	91	1555
FPGA-4	755	8	8	3	143	1	5	280	86	1289
FPGA-5	635	9	9	5	157	1	7	256	158	1237
FPGA-6	2152	10	24	59	170	1	8	326	77	2827
FPGA-7	1723	11	28	77	235	1	9	416	88	2588
FPGA-8	1965	12	16	6	213	1	8	412	72	2705
FPGA-9	2440	15	48	35	261	2	12	498	96	3407
FPGA-10	3291	10	17	42	230	2	11	415	73	4091
FPGA-11	2287	14	38	109	235	1	10	446	127	3267
FPGA-12	3714	14	66	52	240	2	11	440	86	4625
Ratio	0.726	0.004	0.009	0.014	0.072	0.001	0.003	0.136	0.035	1.000

results show that UTPlaceF provides high-quality packing and placement solutions, which outperform the top 3 teams of the ISPD'16 placement contest.

7. REFERENCES

[1] Xilinx, Inc. http://www.xilinx.com.

[2] V. Betz and J. Rose. Cluster-based logic blocks for FPGAs: area-efficiency vs. input sharing and size. In *CICC*, pages 551–554, 1997.

[3] A. S. Marquardt, V. Betz, and J. Rose. Using cluster-based logic blocks and timing-driven packing to improve FPGA speed and density. In *Int'l Symp. on FPGAs*, pages 37–46, 1999.

[4] E. Bozorgzadeh, S. Ogrenci-Memik, and M. Sarrafzadeh. RPack: routability-driven packing for cluster-based FPGAs. In *ASPDAC*, pages 629–634, 2001.

[5] A. Singh, G. Parthasarathy, and M. Marek-Sadowska. Efficient circuit clustering for area and power reduction in FPGAs. *ACM TODAES*, 7(4):643–663, 2002.

[6] H. Liu and A. Akoglu. Timing-driven Nonuniform Depopulation-based Clustering. *Int'l Journal of Reconfig. Comput.*, Article 3, 2010.

[7] S. T. Rajavel and A. Akoglu. MO-Pack: Many-objective clustering for FPGA CAD. In *DAC*, pages 818–823, 2011.

[8] Z. Marrakchi, H. Mrabet, and H. Mehrez. Hierarchical FPGA clustering based on a multilevel partitioning approach to improve routability and reduce power dissapation. In *Int'l Conf. on Reconfig. Comput. and FPGAs*, page 25, 2005.

[9] W. Feng. K-way partitioning based packing for FPGA logic blocks without input bandwidth constraint. In *FPT*, pages 8–15, 2012.

[10] D. T. Chen, K. Vorwerk, and A. Kennings. Improving timing-driven FPGA packing with physical information. In *FPL*, pages 117–123, 2007.

[11] K. Vorwerk and A. Kennings. An improved multi-level framework for force-directed placement. In *DATE*, pages 902–907, 2005.

[12] R. Tessier and H. Giza. Balancing logic utilization and area efficiency in FPGAs. In *FPL*.

[13] M. Tom and G. Lemieux. Logic block clustering of large designs for channel-width constrained FPGAs. In *DAC*, pages 726–731, 2005.

[14] D. Leong M. Tom and G. Lemieux. Un/DoPack: re-clustering of large system-on-chip designs with interconnect variation for low-cost FPGAs. In *ICCAD*, pages 680–687, 2006.

[15] V. Betz and J. Rose. VPR: A new packing, placement and routing tool for FPGA research. In *FPL*, pages 213–222, 1997.

[16] P. Maidee, C. Ababei, and K. Bazargan. Fast timing-driven partitioning-based placement for island style FPGAs. In *DAC*, pages 598–603, 2003.

[17] P. Maidee, C. Ababei, and K. Bazargan. Timing-driven partitioning-based placement for island style FPGAs. *IEEE TCAD*, 24(3):395–406, 2005.

[18] Y. Xu and M. A. S. Khalid. QPF: efficient quadratic placement for FPGAs. In *FPL*, pages 555–558, 2005.

[19] P. Gopalakrishnan, X. Li, and L. Pileggi. Architecture-aware FPGA placement using metric embedding. In *DAC*, pages 460–465, 2006.

[20] M. Xu, G. Grewal, and S. Areibi. StarPlace: A new analytic method for FPGA placement. *Integration, the VLSI journal*, 44(3):192–204, 2011.

[21] M. Gort and J. H. Anderson. Analytical placement for heterogeneous FPGAs. In *FPL*, pages 143–150, 2012.

[22] T. H. Lin, P. Banerjee, and Y. W. Chang. An efficient and effective analytical placer for FPGAs. In *DAC*, page 10, 2013.

[23] Y. C. Chen, S. Y. Chen, and Y. W. Chang. Efficient and effective packing and analytical placement for large-scale heterogeneous FPGAs. In *ICCAD*, pages 647–654, 2014.

[24] ISPD 2016 Routability-Driven FPGA Placement Contest. http://www.ispd.cc/contests/16/ispd2016_contest.html.

[25] T. Lin and C. Chu. POLAR 2.0: An effective routability-driven placer. In *DAC*, pages 1–6, 2014.

[26] M. C. Kim, D. J. Lee, and I. L. Markov. SimPL: An Effective Placement Algorithm. *IEEE TCAD*, 31(1):50–60, 2012.

[27] T. Lin, C. Chu, J. R. Shinnerl, I. Bustany, and I. Nedelchev. POLAR: placement based on novel rough legalization and refinement. In *ICCAD*, pages 357–362, 2013.

[28] W. H. Liu, Y. L. Li, and C. K. Koh. A fast maze-free routing congestion estimator with hybrid unilateral monotonic routing. In *ICCAD*, pages 713–719, 2012.

[29] G. J. Nam, S. Reda, C. J. Alpert, P. G. Villarrubia, and A. B. Kahng. A fast hierarchical quadratic placement algorithm. *IEEE TCAD*, 25(4):678–691, 2006.

[30] T. C. Chen, Z. W. Jiang, T. C. Hsu, H. C. Chen, and Y. W. Chang. NTUplace3: An analytical placer for large-scale mixed-size designs with preplaced blocks and density constraints. *IEEE TCAD*, 27(7):1228–1240, 2008.

[31] S. W. Hur and J. Lillis. Mongrel: hybrid techniques for standard cell placement. In *ICCAD*, pages 165–170, 2000.