Accelerating FPGA Design Space Exploration Using Circuit Similarity-Based Placement

Xiaoyu Shi[1], Dahua Zeng[1], Yu Hu[2], Guohui Lin[1], Osmar R. Zaiane[1]
1. Computing Science Department, University of Alberta
2. Electrical and Computer Engineering Department, University of Alberta

ABSTRACT

This paper describes a novel and fast placement algorithm for field programmable gate array (FPGA) design space exploration. The proposed algorithm generates the placement based on the topological similarity between two configurations (netlists) in the design space. Thus, it utilizes the sharing of reusable information during the design space exploration and avoids the time-consuming placement computation like versatile place and route (VPR). Tested on logic-level and algorithm-level design space exploration cases, our similarity-based placement accurately depicts the "shape" of a design space and pinpoints the designs which are of most interest to IC designers. Moreover, a turbo version of circuit similarity-based placement performs an average of 30x (up to 100x) faster than VPR's while still achieving comparable placement results.

1. INTRODUCTION

An FPGA design offers a variety of customizations by varying design parameters. Those parameters include decisions at the algorithm level (*e.g.*, simple instruction multiple data (SIMD) or pipeline) or at the architecture level (*e.g.*, cache and bus structures); options at high-level synthesis (*e.g.*, scheduling and resource binding tradeoff); combinations of various logic synthesis and optimization (*e.g.*, Berkeley ABC toolset [1]). Efficiency of a design space exploration tool is of paramount importance for designers to quickly identify a small set of favorable design parameter combinations (*i.e.*, configurations) for a multi-objective design. However, FPGA application designers still heavily rely on the general computer-aided design (CAD) tools (*e.g.*, Altera Quartus or Xilinx ISE) to generate every single configuration due to the lack of efficient FPGA design space exploration tools.

Previous work on accelerating design space exploration can be divided into two categories, (1) methods that minimize the number of configurations to be evaluated [6], (2) methods that generate design evaluation by modeling [9] [4] [12]. The runtime for generating (*i.e.*, synthesizing) one configuration is crucial for the efficiency of both methodologies. In this paper, we accelerate placement, one of the most time-consuming phases in FPGA synthesis, to increase the efficiency of generating each configuration in the design space.

A unique property of design space exploration problem is the existence of *similarities* between netlists of different configurations. Such similarities include both *local similarity* and *global similarity*. The local similarity is due to the use of common primitives (*e.g.*, digital signal processing (DSP) modules or macros) in different configurations. The global similarity exists because the characteristics of the application are shared by all configurations that implement it. This property is illustrated by an example of an algorithm-level design space exploration problem with two implementation algorithms (*i.e.*, RAG-n [3] and Hcub [20]) of a constant multiplier block. The algorithm-level schematics of these two implementations (generated by CMU SPIRAL [2]) are shown in Figure 1. They both implement the following constant multiplier block:

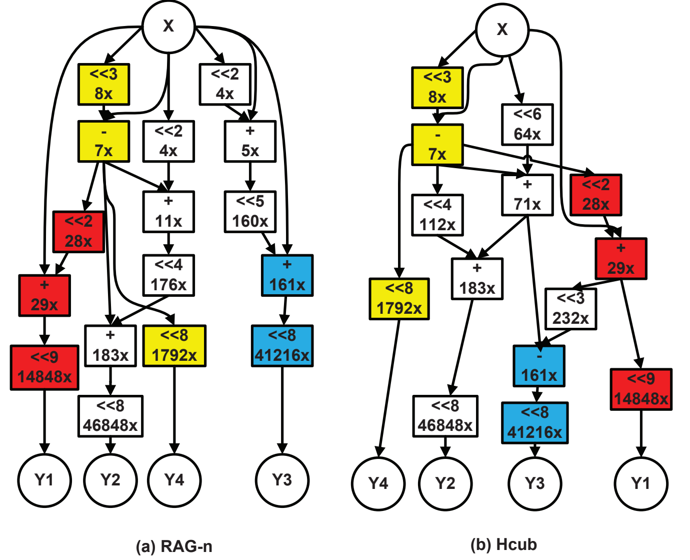


Figure 1: Constant multiplier blocks generated by CMU SPIRAL (integer constants: 58, 183, 161, 7; bit width is 8)

$$Y_1 = 58 \cdot X, Y_2 = 183 \cdot X, Y_3 = 161 \cdot X, Y_4 = 7 \cdot X \qquad (1)$$

where X is the input and $Y_1, \cdots, Y_4$ are outputs, and the precision (bit width) is 8 bits. Although there is a significant difference between the structures of these two configurations at the first glance, they both use adders, subtractors and shifters as building blocks (primitives), which lead to a local similarity. When these algorithm level designs are mapped to FPGAs, such local similarity results in similarities of local clusters that contain look-up tables (LUTs) or DSPs used to implement these primitives. In addition, both configurations generate the constant multiplication for equation (1), which results in global similarity. Specifically, the I/O (X and Ys) of both configurations are identical; there are identical internal structures (*e.g.*, subgraph $28x \rightarrow 29x \rightarrow 14848x$ is shared by both implementations) as highlighted in Figure 1 using different colors; both configurations are sparse directed acyclic graph (DAG) structures, and they are topologically similar.

Our approach takes advantage of such properties of the design

space exploration problem to accelerate the process of generating placements for all configurations in the design space. In a nutshell, our similarity-aware placement starts with the computation of a reference placement for a particular configuration using an existing FPGA placer (*e.g.*, VPR). Once this reference placement is generated, the placements for the rest of configurations in the design space are generated very efficiently based on this reference placement by exploiting the similarity property. Figure 2 shows the CAD flow of the design space exploration with similarity-aware placement.

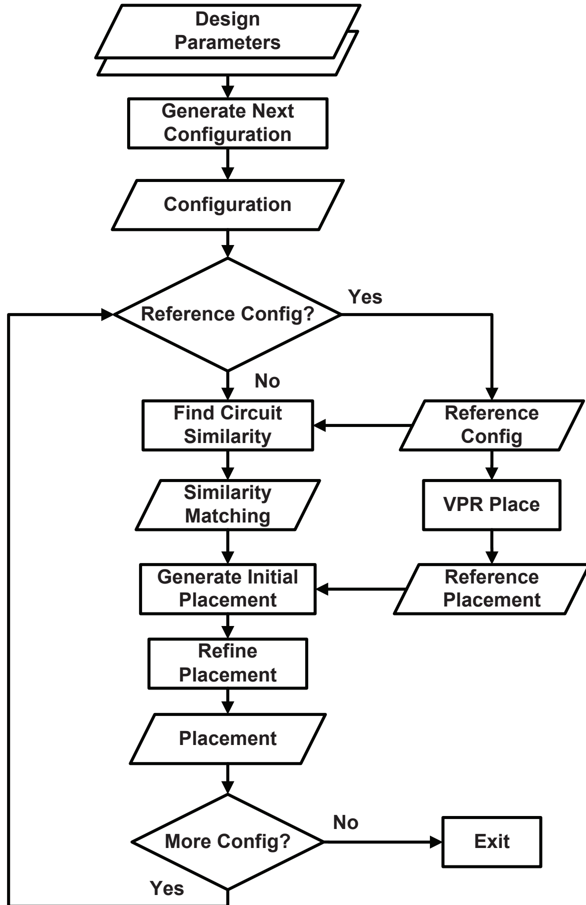


Figure 2: CAD flow for the design space exploration using circuit similarity-based placement

The kernel of the proposed placement is the *circuit similarity* algorithm, which performs a fast and automatic detection of the similarity (both local and global) between a new configuration and the reference configuration. The similarity is detected based on the topological structures between these two netlists and corresponding node matchings are obtained. Based on the detected circuit similarity, we then generate an initial placement for the new configuration. A placement refinement is finally applied to the initial placement results.

To verify the effectiveness of the proposed similarity-aware placement for accelerating design space exploration, we have performed experiments in two design space exploration problems, at the logic-level and algorithm-level, respectively. In both cases, experimental results show that our circuit similarity-based placement captures the characteristics of the design space with accurately estimated wire length and critical delay, and pinpoints the best designs. Moreover, our approach achieves averaged 30x speedup compared to VPR's placement.

The remainder of this paper is organized as follows. Section 2 illustrates the overall circuit similarity-based placement CAD flow for design space exploration with an example. Section 3 provides preliminaries for this paper. Section 4 describes the circuit similarity algorithm. Section 5 experimentally demonstrates logic-level and algorithm-level design space exploration. The paper is concluded in Section 6.

2. MOTIVATING EXAMPLE

In this section, we illustrate the proposed similarly-aware placement for a logic-level design space exploration problem. The design parameters explored in this example are the logic synthesis options in Berkeley ABC tool set [1]. The purpose of the design exploration is to identify the impact of different combinations of logic optimizations on the wire length and timing. MCNC benchmark [21] "des" is used as the application to be implemented. Suppose the reference configuration is synthesized by the following ABC script:

b; rs; rs -K 6; b; rsz; rsz -K 6; b; rsz -K 5; b

where each command is a logic optimization in ABC, *e.g.*, "b" means balance the and-inverter graph and "rs" means the logic rewriting using Boolean substitution. The placement of the reference configuration is generated by VPR and the layout is shown in Figure 3(a) with nets toggled in VPR GUI. Next we generate a new configuration, which is synthesized by the following ABC script:

st; rw -l; b -l; rw -l; rf -l; fraig; rw -l; b -l; rw -l; rf -l

The similarity of the netlists for the reference and the new configuration is obtained by a circuit similarity algorithm (detailed in Section 4). Based on this similarity, the layout of the initial placement is shown in Figure 3(b), which is obtained based on the placement of the reference configuration. As circled in the figures, the initial placement using circuit similarity captures the main characteristics of the topological similarity between the reference and the new configurations, and thus results in a well optimized initial layout. Given this initial placement, a low-temperature annealing process is used to refine the placement and final placement result is shown in Figure 3(c). For comparison, Figure 3(d) shows the layout of the random initial placement produced by VPR. Obviously, the initial placement generated by circuit similarity has significantly less wire length compared to the random placement. This shows that the circuit similarity algorithm successfully finds the internal corresponding topologies of both netlists, and therefore makes a good decision on the relative position of clustered logic blocks (CLBs) to be placed. It is also interesting to compare the highlighted topologies of the layouts of the final placement between the new configuration and the reference configuration (Figure 3(c) and Figure 3(a)). Although different logic optimizations are applied, the resulting layout shares a similar layout. The final placement produced by VPR is also shown in Figure 3(e) for comparison. Table 1 shows the numerical comparisons of these layouts, where delay cost quantifies the delay of a route from a net source to any of its sinks. The placement generated by our similarity-aware algorithm results in comparable wire length, better critical path delay and less placement time compared with the placement generated by VPR.

3. PRELIMINARIES

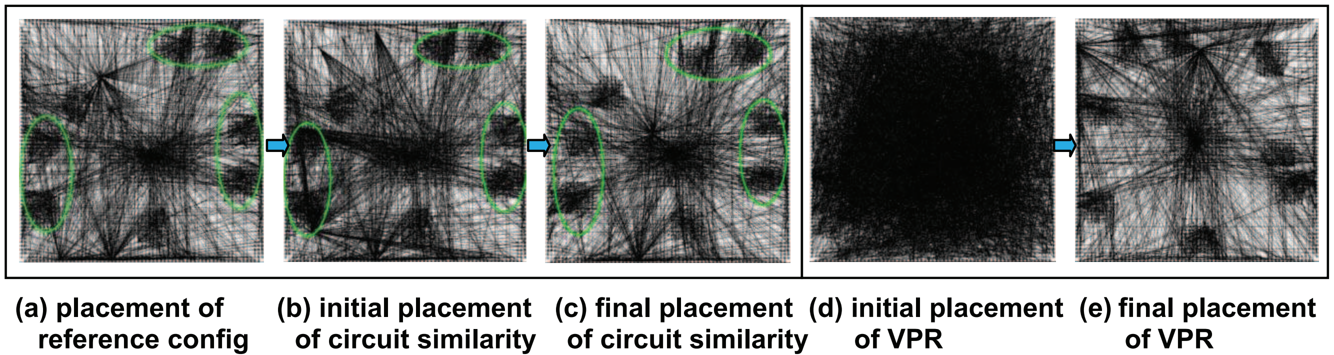


Figure 3: Placement results for circuit "des". (reference configuration has 1245 CLBs and 1501 nets, the new configuration has 1215 CLBs and 1471 nets)

Table 1: Status of layouts of Figure 3

Layout	Wire	Delay cost	Critical delay	Runtime (s)
CS-init	306	5.93E-05	-	-
VPR-init	1087	1.40E-04	-	-
CS-final	237	5.08E-05	8.28E-08	13.38
VPR-final	221	4.98E-05	1.01E-07	28.42

3.1 Design Space Exploration

For multi-objective optimization, a pareto-optimal point represents a design point or a configuration for which no other configuration is better in the objective function space. A common goal of design space exploration in multi-objective optimization is to find pareto-points, which benefit designers seeking to make appropriate design tradeoffs for given constraints. Pareto-point exploration is time-consuming due to the exponential increase of the number of configurations w.r.t. the tuneable parameters and the long runtime required by the current FPGA CAD tools. For example, current configurable soft cores (*e.g.*, Xilinx Microblaze cores) can have thousands of configurations [16] and the runtime for evaluating one configuration of Microblaze by Xilinx platform studio is about 15 minutes [6]. Obviously, a straightforward evaluation of all configurations for pareto-points is infeasible.

Besides pareto-points, which only characterize a set of "good configurations" in the design space, people may also be interested in finding the "shape" of the entire design space, which includes both "good configurations" and "bad configurations". The knowledge of "bad configurations" is helpful for algorithm developers to diagnose the design and for CAD tool designers to analyze the tool flow. For instance, the combination of two logic optimizations may result in a worse design than applying them individually. A full profiling of the design space will reveal such phenomenon and help CAD tool designers improve the tool.

3.2 Graph Similarity

Given two graphs or networks, there are multiple ways to define their similarity. The isomorphism method identifies a bijection between the nodes of two graphs which preserves (directed) adjacency [13]. The edit distance method determines the minimum cost transformation from one graph to another by giving a cost function on edit operations (*e.g.* addition/deletion of nodes and edges) [5]. The common subgraph method identifies the "largest" isomorphic subgraphs of two graphs [10]. These methods all consider global topological information. However, the time complexity of these methods is NP-hard. Statistical methods provide linear time complexity by assessing aggregate measures of graph structure (*e.g.* degree distribution, diameter, betweenness measures) but lack of global topological information[14]. Therefore, in order to take consideration of both time complexity and global topological information, our circuit similarity algorithm employs iterative methods, which are based on the theory that two graph elements (*e.g.*, edges or nodes) are similar if their neighborhoods are similar [19].

Different algorithms, including similarity flooding [15], simRank [7], and the coupled node-edge algorithm [22], have been proposed to compute graph similarity based on the iterative definition. For our algorithm, we use an iterative graph similarity algorithm for molecular graphs [11], which takes advantage of the graph sparsity, one of the properties of a circuit graph. Before presenting this algorithm, Table 2 describes all frequently used variables.

Table 2: Summary of variables in iterative similarity algorithm

Variable	Description
$X_{i,j}^{(n)}$	Similarity score between node i in graph $G(V)$ and node j in graph $G'(V')$ in iteration n
v_i	A node in graph G
v'_j	A node in graph G'
t	The number of iterations
$n(v)$	The set of all adjacent nodes of node v
π	An injective map from $n(v_i)$ to $n(v'_j)$, if $\lvert n(v_i)\rvert < \lvert n(v'_j)\rvert$ An injective map from $n(v'_j)$ to $n(v_i)$, if $\lvert n(v_i)\rvert \geq \lvert n(v'_j)\rvert$
α	A weight constant within interval (0,1)
ϵ	A terminating threshold for iterations
M	An upper bound for number of iterations
$k_v : V \to V'$	A predefined inter-similarity between two nodes
$k_e : E \to E'$	A predefined inter-similarity between two edges, where (v_i, v) is an edge in graph G and $(v'_j, \pi(v))$ is an edge in graph G'
$in(v)$	The set of all adjacent nodes that have an edge entering node v
$out(v)$	The set of all adjacent nodes that have an edge leaving node v

The iterative similarity algorithm is summarized in Algorithm 1. In each iteration, the algorithm computes the *similarity score*, $X_{i,j}^{(t)}$, between each node pair (v_i, v'_j) in two graphs, where $v_i \in G$ and $v'_j \in G'$. The similarity score of a node pair is a real value between 0 and 1. The higher the similarity score of a node pair is, the more likely these two nodes are matched together. This score is updated based on the values of their adjacent node pairs obtained in the previous iteration and the predefined inter-similarity

between two nodes/edges. The predefined similarity is used to capture non-topological connections between two graphs. The algorithm terminates when the difference between the total similarity scores in two consecutive iterations is smaller than ϵ, or the number of iterations reaches an upper bound M.

Algorithm 1 Similarity of G and G'

Initialize $X_{i,j}^{(0)}$
while $|\sum X^{(t)} - \sum X^{(t-1)}| > \epsilon$ and $t < M$ **do**
  if $|n(v_i)| < |n(v_j')|$ **then**

$$X_{i,j}^{(t)} = (1-\alpha)k_v(v_i, v_j^{'}) + \alpha \max_{\pi} \frac{1}{|n(v_j^{'})|} \sum_{v \in n(v_i)} X_{v,\pi(v)}^{(t-1)} k_e((v_i, v), (v_j^{'}, \pi(v)))$$

  else

$$X_{i,j}^{(t)} = (1-\alpha)k_v(v_i, v_j^{'}) + \alpha \max_{\pi} \frac{1}{|n(v_i)|} \sum_{v^{'} \in n(v_j^{'})} X_{\pi(v^{'}),v^{'}}^{(t-1)} k_e((v_i, \pi(v^{'})), (v_j^{'}, v^{'}))$$

4. CIRCUIT SIMILARITY

4.1 Circuit Similarity Detection

Algorithm 1 is designed for undirected molecular graphs [11] where the computational complexity is too expensive to handle real circuits. In this subsection, we first adapt Algorithm 1 to consider a directed circuit graph and then present two techniques to significantly improve both time and space efficiency of the circuit similarity detection.

One unique constraint for circuit similarity detection is that the matching of the corresponding primary inputs (PIs) and primary outputs (POs) of the two circuits must be guaranteed. Therefore, the similarity score for a pair of corresponding PI/PO nodes is set to be constant 1 and is not updated during the iteration. As a result, such a predefined PI/PO matching effectively provides extra hints for the iterative similarity detection process and generates better matching between the two circuits. Intuitively, for those node pairs close to PI/PO nodes, higher scores will be obtained because of the propagation of the constant similarity score set in PI/PO node pairs. Note that other hints such as internal registers and naming matching information obtained in logic synthesis can also be used as predefined matching to enhance both the quality and speed of the circuit similarity detection.

For those internal nodes without predefined similarity, we replace k_v with $X_{i,j}^{(t)}$, and k_e with 1. Instead of updating similarity scores based on all the neighbors, we can perform the update for edges that leave the nodes and edges that enter the nodes, separately. More specifically, given the two graphs, we initialize the similarity scores of all pairs of nodes to be 1. In each iteration, for $|in(v_i)| < |in(v_j^{'})|$ and $|out(v_i)| < |out(v_j^{'})|$, the update of similarity score $X_{i,j}^{(t)}$ is modified as follows

$$X_{i,j}^{(t)} = (1-\alpha)X_{i,j}^{(t-1)} + \alpha \frac{1}{|out(v_i)| + |in(v_i)|} [\max_{\pi}(\sum_{v^{'} \in out(v_j^{'})} X_{\pi(v^{'}),v^{'}}^{(t-1)}) + \max_{\pi}(\sum_{v^{'} \in in(v_j^{'})} X_{\pi(v^{'}),v^{'}}^{(t-1)})]$$

In our experiment, we find $\alpha = 0.75$ gives the best matching quality. After obtaining a similarity matrix that describes a complete bipartite graph, where the weight associated with each edge denotes the similarity score of two nodes, we can then compute a maximum matching in this bipartite graph to obtain a node matching between the two graphs. The min-cost network flow [17] is used to compute the maximum matching in our experiment.

4.2 Performance Enhancement

In practice, it is infeasible to compute the similarity scores of all $|V| \cdot |V'|$ node pairs for large circuits. In this subsection, we present two pruning techniques for DAGs to reduce the number of pairs that need to be updated so that we can reduce both the runtime and storage requirement.

Support Constraint. Two internal nodes are less likely to be matched if they share few predefined matchings in their supports. A *support* of a node is the set of nodes with predefined matchings in the transitive fanin or fanout cone of this node. Formally, for two nodes $v \in G$ and $v' \in G'$, the support constraint requires

$$\min(\frac{X_{SP(v),SP(v')}}{|SP(v)|}, \frac{X_{SP(v),SP(v')}}{|SP(v')|}) \geq \beta$$

where $X_{SP(v),SP(v')}$ denotes the *support similarity* of v and v', which is the sum of similarity scores of all $v \to v'$ node pairs in their supports, $SP(v)$ is the support node set of v and $\beta \in (0, 1]$ is a constant. Likely, two nodes with higher supports tend to be matched together. If the support constraint of the two nodes is not satisfied, we do not update their similarity score in the iteration. For example, if $\beta = 1$, *i.e.*, we only keep the pairs of nodes that have exactly the same supporting PIs and POs.

Level Constraint. If only combinatorial resynthesis is involved, we can convert a circuit into a DAG by removing all registers and adding the register inputs (outputs) as POs (PIs). Given a DAG, a topological sort and reverse topological sort can label each internal node v with two values, *i.e.*, $level(v)$ and $rlevel(v)$, where $level(v)$ ($rlevel(v)$) denotes the length of the longest path from PIs (node v) to node v (POs). Two nodes with significantly different ($level$, $rlevel$) values are less likely to be matched. Formally, for two nodes $v \in G$ and $v' \in G'$, the level constraint requires

$$|level(v) - level(v')| \leq B_l, |rlevel(v) - rlevel(v')| \leq B_r$$

where B_l and B_r are two nonnegative constant integers. For example, if B_l and B_r are both set to be zero, we only keep the pairs of nodes that are on the exact same level.

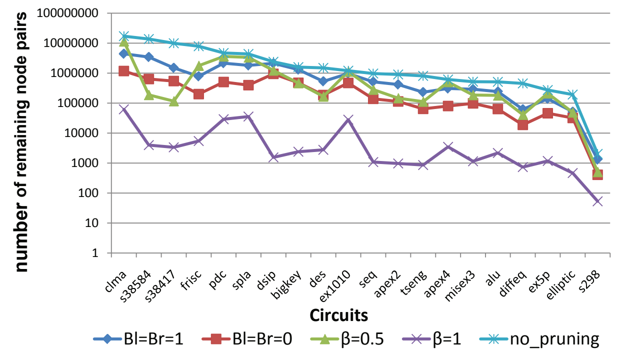


Figure 4: Effectiveness of the present pruning techniques

We have tested the above two pruning techniques on the MCNC

benchmarks. For each circuit, we run two logic synthesis algorithms (one with ABC command “if -k 4” and the other with “if -k 4; imfs”) and generate two logic-level netlists. Figure 4 compares the total number of node pairs that updated in the iterative algorithm with the following five schemes: (a) without pruning (“no_pruning”), (b) using a weak level constraint-based pruning (“B_l=B_r=1”), (c) using a strong level constraint-based pruning (“B_l=B_r=0”), (d) using a weak support constraint-based pruning (“β=0.5”), and (e) using a strong support constraint-based pruning (“β=1”). As shown in Figure 4, our pruning techniques reduce the number of node pairs by three to four orders of magnitude compared with the total number of node pairs. More specifically, the strong level constraint-based pruning (“B_l=B_r=0”) and the strong support constraint-based pruning (“β=1”) can prune around 90% and 99% node pairs, respectively. As a result of the sparsity of the similarity matrix, the maximum matching algorithm is significantly faster. In Section 5, we will show that these pruning techniques do not significantly degrade the quality of the similarity detection and node matching when we apply the circuit similarity-based placement on design space exploration.

4.3 Circuit Similarity-based Placement

As shown in Figure 2, circuit similarity is used to speed up placement which allows faster design space exploration. More specifically, given a network G where each node denotes a LUT and each edge denotes an interconnection between LUTs, the placement of network G can be obtained by performing a highly-optimized placement (*e.g.*, VPR [18]). For another network, G', which is generated by other design parameters, its placement is generated by first computing the similarity between networks G and G', and finding the correspondence of nodes in these two networks. Based on such node correspondence, the initial placement of network G' can be determined using the placement of network G, *e.g.*, if node V' in network G' corresponds to node V in network G, V' is assigned the same coordinates as node V. Further refinement (*e.g.*, low-temperature simulated annealing) is applied to the initial placement of G' to gain better results.

5. DESIGN SPACE EXPLORATION

5.1 Logic-Level Design Space Exploration

5.1.1 Experimental CAD Flow

The objective of this design space exploration is to identify the influence of logic-level optimization to a post-layout design. Following Figure 2, the design parameters are logic synthesis and optimization commands in Berkeley ABC [1]. Although there are many possible combinations and execution sequences of those commands, in our experiment, we use 19 synthesis scripts provided in abc.rc from the ABC package, *i.e.*, there are 19 configurations in this design space exploration case. In the rest of this section, we follow the same names of each script used in ABC as the index, *e.g.*, the two scripts shown in Section 2 are named “resyn3” and “rwsat2", respectively. Interested readers may refer to the “abc.rc” file provided with the ABC download for details.

The experimental CAD flow is shown in Figure 5. Starting from 19 ABC logic synthesis scripts, we have the resulting synthesized netlists stored in BLIF file format. Next, a technology mapping (using ABC command “if -k 4”) is performed on the netlists to map them into a 4-LUT-based network. Afterwards, the technology mapped netlists are packed into CLBs using T-VPack [18] with “no_cluster” option, where each CLB contains one LUT and one flip-flop. After this point, we compare two CAD flows: (a) circuit similarity-based flow and (b) VPR *from-scratch* flow, as shown in Figure 5. Flow (a) first selects the largest configuration (*i.e.*, the one with largest number of CLBs) as the reference. Then the reference configuration is placed using VPR and produces a reference placement (“.p” file). The reference configuration and its placement are then used to guide the initial placement of the new configuration by finding the similarity between the new configuration and the reference configuration. A low-temperature annealing process using VPR (initial temperature is set to 0.1) is performed to further refine the placement results. Flow (b) simply uses VPR to re-place every single configuration from scratch.

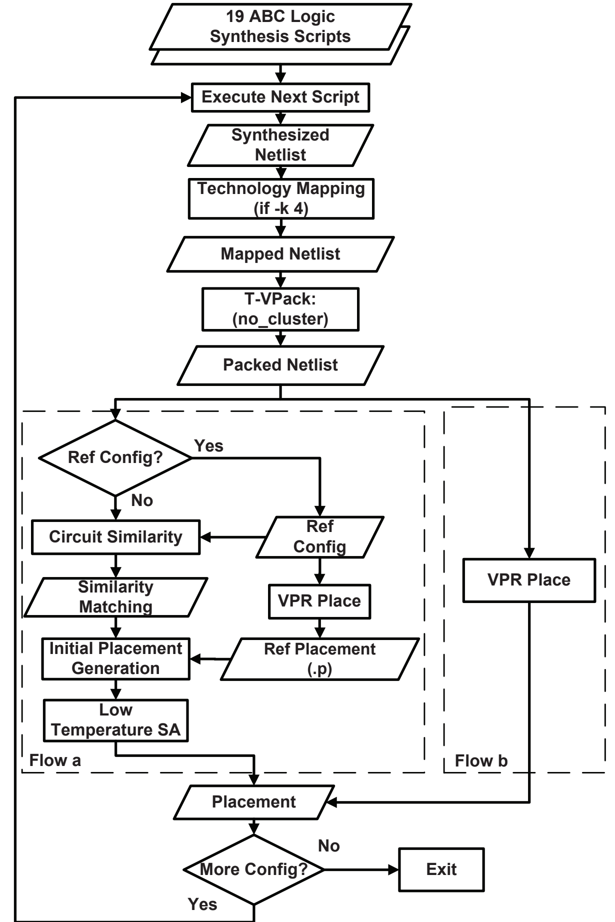


Figure 5: CAD flows used in the experiments for design space exploration

5.1.2 Experimental Settings

As stated in Section 4, based on different pruning settings, we develop two versions of circuit similarity. A high-quality version, CS, uses $\beta = 0.5, B_l = B_r = 1$ and $inner_num = 1$[1]. A turbo version, CS-t, uses $\beta = 1, B_l = B_r = 0$ and $inner_num = 0.1$. Both CS and CS-t are evaluated in our experiments.

Our proposed circuit similarity algorithm is implemented in C and evaluated on the 20 applications from the MCNC benchmark. We collect the results on a Linux server with an 8-core 2.66GHz CPU and 32GB memory averaged over five runs. The CS2 package

[1]A factor in VPR which controls the number of moves at each temperature.

[8] is used to solve the min-cost network flow for the maximum matching problem in the circuit similarity algorithm.

5.1.3 *Experimental Results*

Table 3 shows the minimal and maximal CLB number and level for the design space of each application. The number of CLBs and levels vary widely in different configurations.

Table 3: Characteristics of the logic-level design space for 20 MCNC applications over 19 configurations

			block#		Level	
Circuit	CI#	CO#	min	max	min	max
alu4	14	8	652	710	7	10
apex2	38	3	773	926	8	11
apex4	9	19	754	805	7	10
bigkey	452	421	924	1263	3	4
clma	94	115	3731	4221	12	17
des	256	245	1157	1245	6	10
diffeq	332	308	648	712	12	15
dsip	452	421	1106	1554	3	4
elliptic	196	196	375	441	8	11
ex1010	10	10	851	1100	6	10
ex5p	8	63	460	519	6	11
frisc	905	1002	2173	2788	19	26
misex3	14	14	545	680	6	9
pdc	16	40	1836	2159	8	14
s298	17	20	36	43	3	4
s38417	1490	1568	3063	3252	9	10
s38584	1297	1564	3568	3715	8	11
seq	41	35	891	982	6	9
spla	16	46	1718	2074	8	14
tseng	435	507	744	938	12	14

Quality of the initial placement. Table 4 shows the initial placement quality of CS and CS-t compared to VPR's initial results. Due to limited space, we show one representative circuit, "dsip" as an example. The results for the other circuits are similar. The "Configuration" column in Table 4 lists the 19 ABC scripts' names. Two essential measures in initial placement stage are compared, the bounding box cost ("initial bb cost" column) and the delay cost ("initial delay cost" column). The initial placement results generated by CS and CS-t are significantly better than VPR's random initial placement results. CS improves the bb cost and delay cost by 76% and 48% compared to VPR, respectively. This demonstrates that circuit similarity algorithm indeed discovers the intrinsic structural connections among different configurations, and thus provides a quality placement for the design space exploration.

Quality of the final placement. A low-temperature annealing is applied to the initial placement results generated by our circuit similarity-based placement. Table 5 compares the final placement results of circuit "dsip" for 19 designs. We evaluate the final wire length and the critical delay. For wire length, CS and CS-t produce the results close to VPR's final results with 32% and 53% overhead, respectively. For critical delay, CS and CS-t achieve better results than VPR, reducing it by 18% and 20%, respectively. This shows the effectiveness of circuit similarity-based placement that generates an optimized initial placement which in turn leads to an optimized final placement. The comparison between CS and CS-t in Table 5 proves the effectiveness of the proposed pruning techniques as well (in Section 4). CS-t, geared with aggressive pruning and significantly lower annealing effort, still produces placement with comparable or even better quality than VPR.

Design space shape characterization. We compare the

Table 4: Initial placement quality comparison of circuit "dsip" for 19 designs

	initial bb cost			initial delay cost		
Configuration	CS	CS-t	VPR	CS	CS-t	VPR
resyn	363	934	1817	1.02E-04	1.49E-04	2.22E-04
resyn2	363	934	1819	1.02E-04	1.49E-04	2.19E-04
resyn2a	453	453	2184	1.69E-04	1.69E-04	3.11E-04
resyn3	443	494	2189	1.64E-04	1.70E-04	3.07E-04
compress	448	448	2203	1.66E-04	1.66E-04	3.31E-04
compress2	363	936	1785	1.03E-04	1.52E-04	2.21E-04
choice	1943	1978	2020	2.72E-04	2.74E-04	2.72E-04
choice2	1940	1977	1990	2.72E-04	2.74E-04	2.75E-04
rwsat	371	930	1804	1.03E-04	1.53E-04	2.26E-04
rwsat2	448	578	2184	1.64E-04	1.75E-04	3.30E-04
shake	424	518	2151	1.44E-04	1.51E-04	2.93E-04
share	365	936	1778	1.02E-04	1.50E-04	2.19E-04
src_rw	458	793	2185	1.48E-04	1.70E-04	2.96E-04
src_rs	452	458	2159	1.46E-04	1.48E-04	2.96E-04
src_rws	544	841	2172	1.53E-04	1.72E-04	2.93E-04
resyn2rs	365	937	1770	1.03E-04	1.52E-04	2.23E-04
compress2rs	363	936	1809	1.03E-04	1.53E-04	2.22E-04
resyn2rsdc	427	719	2156	1.65E-04	1.83E-04	3.10E-04
compress2rsdc	373	933	1772	1.04E-04	1.53E-04	2.24E-04
geomean	483	802	1989	1.39E-04	1.69E-04	2.65E-04
ratio	24%	40%	1	52%	64%	1

minimal, median and maximal wire length and critical delay produced by CS and CS-t to VPR. Figure 6 shows the minimal critical delay curves of all 19 designs for 20 circuits using CS, CS-t and VPR. The almost identical curves prove that both CS and CS-t can precisely pinpoint the minimal critical delay design. Due to limited space, we are unable to show the curves of the median and maximum. Nevertheless, the curves of both CS and CS-t follow close to VPR's. Moreover, the shape of most configurations is accurately matched as well. As an example, Figure 7 shows the wire length curve for circuit "dsip". Note that "choice" and "choice2" include a repetitive call of ABC synthesis command "fraig_store", which stores the current network as one "synthesis snapshot" for later technology mapping. Such a choice-based logic optimization may significantly change the topology of the netlist. In the future, we will investigate improvements to this problem. Although CS-t produces longer wire length than VPR, Figure 7 shows that CS-t closely captures the relative wire length of each configuration which is essentially useful in the design space exploration.

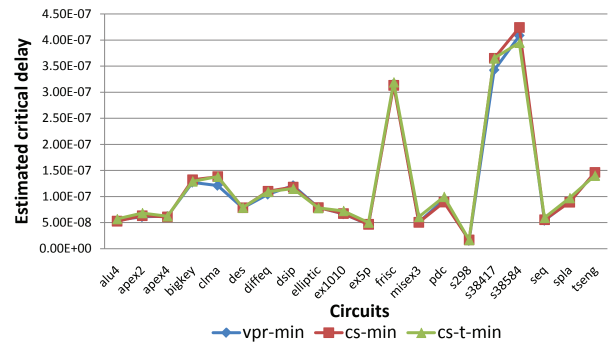


Figure 6: Minimal estimated critical delay design space shape of 20 circuits on 19 designs

Runtime Comparison. Table 6 compares the total runtime[2]

[2]Note that the '*' marked time is measured when timeout

Table 5: Final placement quality comparison of circuit "dsip" for 19 designs

	Wire length			Critical delay		
Configuration	CS	CS-t	VPR	CS	CS-t	VPR
resyn	353	432	267	1.23E-07	1.18E-07	1.63E-07
resyn2	353	432	278	1.26E-07	1.17E-07	1.51E-07
resyn2a	465	523	432	1.25E-07	1.25E-07	1.34E-07
resyn3	464	516	424	1.25E-07	1.22E-07	1.28E-07
compress	465	519	400	1.25E-07	1.19E-07	1.63E-07
compress2	354	434	231	1.19E-07	1.18E-07	1.81E-07
choice	722	735	369	1.30E-07	1.30E-07	1.27E-07
choice2	721	732	346	1.30E-07	1.30E-07	1.54E-07
rwsat	351	433	281	1.21E-07	1.15E-07	1.40E-07
rwsat2	468	518	426	1.26E-07	1.22E-07	1.48E-07
shake	440	507	370	1.20E-07	1.21E-07	1.77E-07
share	352	431	236	1.22E-07	1.18E-07	1.64E-07
src_rw	451	512	388	1.24E-07	1.25E-07	1.72E-07
src_rs	445	502	394	1.21E-07	1.22E-07	1.24E-07
src_rws	441	514	411	1.25E-07	1.22E-07	1.65E-07
resyn2rs	352	429	225	1.18E-07	1.16E-07	1.63E-07
compress2rs	352	435	242	1.24E-07	1.19E-07	1.73E-07
resyn2rsdc	462	516	412	1.23E-07	1.24E-07	1.21E-07
compress2rsdc	352	423	216	1.22E-07	1.16E-07	1.53E-07
geomean	429	496	324	1.24E-07	1.21E-07	1.52E-07
ratio	132%	153%	1	82%	80%	1

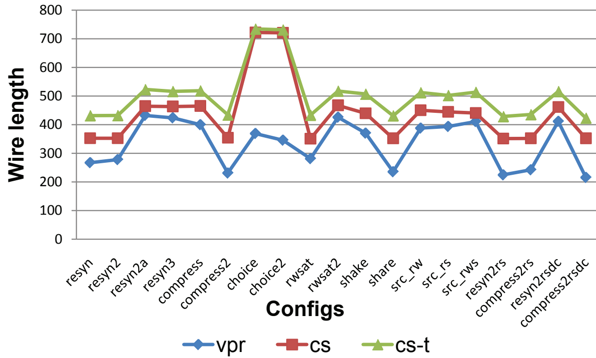


Figure 7: Final wire length design space shape comparison of VPR, CS and CS-t on circuit "dsip"

of placing 19 designs of each MCNC application using CS, CS-t and VPR. Column "Ref" shows the time to place the reference configuration. Column "CS" and "CS-t" shows the time to generate the placement for the rest of 18 configurations. Without considering the time for placing the reference configuration, CS achieves averaged 3x speedup while CS-t achieves averaged 30x speedup with up to 100x compared to from-scratch VPR placement. Since the computation of the reference placement is a one-time cost for a design space exploration problem of one application, the time used for the reference placement should be amortized and becomes negligible as the number of configurations increases.

In practice, one can take advantage of the significant speedup of CS-t and use it to perform quick design space exploration. For instance, the total time of exploring the whole design space of 20 MCNC applications with 19 designs is more than 8 hours using VPR. In contrast, it only takes 37 minutes (including the time for the reference placement) using our CS-t. More significant speedup is expected when larger design space is explored.

is invoked. If CS takes longer than the placement time of the reference configuration, a timeout is invoked and it stops the program from running.

Table 6: Comparison of total runtime (s) for logic-level design space exploration. The '*' marked time is measured with a timeout

Circuit	CS	CS-t	VPR	Ref
alu4	113.99 (2x)	6.13 (31x)	188.62	10.37
apex2	122.85 (2x)	7.7 (35x)	267.46	17.03
apex4	187.18* (1x)	8.77 (28x)	246.03	14.89
bigkey	630.69 (1x)	26.71 (27x)	720.96	37.85
clma	1782.7* (1x)	101.42 (25x)	2532.65	143.91
des	183.37 (3x)	20.32 (27x)	549.24	29.38
diffeq	56.09 (7x)	9.13 (42x)	387.89	19.78
dsip	626.48* (1x)	55.92 (14x)	776.66	47.12
elliptic	47.06 (4x)	4.37 (40x)	173.6	9.88
ex1010	229.28* (1x)	72.73 (5x)	336.51	18.54
ex5p	95.55* (1x)	4.18 (33x)	136.49	7.87
frisc	373.98 (8x)	75.43 (42x)	3178.44	176.56
misex3	122.87 (1x)	4.68 (36x)	170.32	11.44
pdc	747.53* (1x)	43.8 (22x)	975.6	55.11
s298	0.94 (5x)	0.32 (14x)	4.45	0.27
s38417	564.45 (15x)	82.99 (100x)	8318.7	445.85
s38584	541.19 (15x)	84.73 (96x)	8155.16	443.59
seq	218.33 (2x)	9 (37x)	337.44	18.52
spla	745.35* (1x)	39.71 (23x)	923.75	53.36
tseng	86.72 (7x)	13.71 (43x)	587.89	33.35
geomean	186.97 (3x)	16.64 (30x)	497.81	28.33
total	7476.6 (4x)	671.75 (43x)	28967.86	1594.67

5.2 Algorithm-Level Design Space Exploration

5.2.1 *Experimental CAD Flow and Settings*

We now demonstrate the effectiveness of the proposed placement at the algorithm-level design space exploration. The design is a constant multiplier, where a multiplier block implements a parallel multiplication of a variable with a fixed set of constants, *i.e.* $c_1, c_2, \ldots c_n$. The design parameter in this exploration is the fractional bits, which controls the precision of the constants, from 7 to 25, resulting in a design space containing 18 configurations[3]. Given a fractional bit setting, we use CMU SPIRAL multiplier block generator to generate the register transfer level (RTL) design of each configuration based on the Hcub algorithm [20]. The following constants (accurate to two decimal places) are used for all configurations: 0.23, 0.71, 0.63, 0.03, -0.19, 0.03, 0.03, -0.01. Once we obtain the RTL design, we use Altera Quartus to perform RTL elaboration and generate a BLIF file from a verilog (.v) file. Other experimental settings are the same as described in Section *5.1.2*. Table 7 presents the number of CLBs and level for algorithm-level design space. Since those configurations vary in algorithm level, the topological structure and circuit size differ considerably compared to logic-level variations. Therefore, it is more challenging to find the similarities between these design configurations.

5.2.2 *Experimental Results*

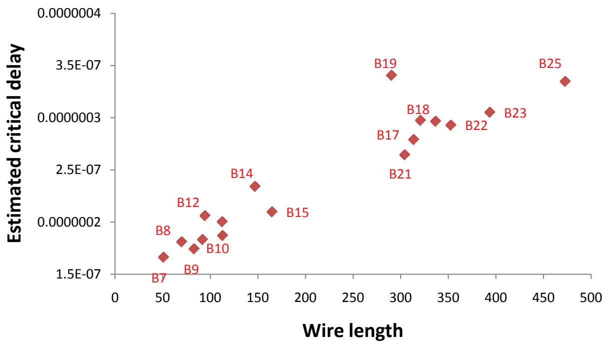
Figure 8(a) shows the wire length-critical path delay space produced by CS and VPR-based placement for the 18 configurations. The label besides each point indicates the corresponding configuration. For example, "B7" means this point corresponds to the configuration using $Bits = 7$. Figure 8(b) shows the same design space using CS. From these two figures, we can clearly see that CS and VPR find the same pareto-points, *i.e.*, optimal configurations of this design space, such as $B7$, $B8$ and $B9$. In addition, the overall shapes of the two design spaces match well. This proves

[3] $Bits = 16$ is abandoned since ABC crashed when synthesized it. So there are 18 configurations in total.

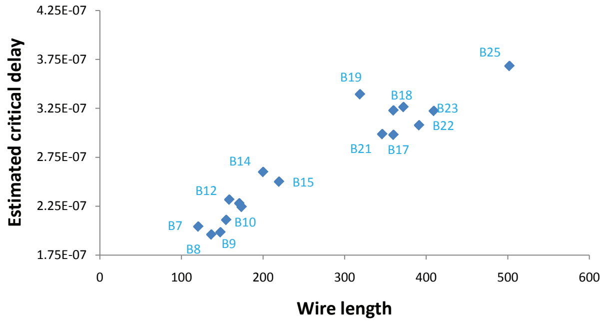
Table 7: Characteristics of the algorithm-level design space of 18 configurations using CMU SPIRAL

Bits	CLB#	Level	*Bits*	CLB#	Level
7	501	35	*17*	2222	57
8	697	38	*18*	2356	52
9	814	37	*19*	2339	60
10	920	41	*20*	2577	56
11	1115	42	*21*	2398	54
12	938	43	*22*	2625	55
13	1085	41	*23*	2832	55
14	1293	48	*24*	3289	56
15	1352	48	*25*	3234	62

that our circuit similarity not only works well at low level logic synthesis, but also at high level algorithm level. Moreover, in terms of runtime, CS and CS-t achieve 7x and 30x speedup compared to VPR, respectively.



(a) Wire length-delay space of VPR for 18 configurations



(b) Wire length-delay space of CS for 18 configurations

Figure 8: Comparison of wire length-delay space of VPR and CS

6. CONCLUSIONS AND FUTURE WORK

In this work, we have presented our proposed circuit similarity-based placement for accelerating FPGA design space exploration. The characteristics of each design can be automatically captured by finding the similarity between each configuration and a reference configuration. The experimental results prove that our circuit similarity works well at both logic level and algorithm level. The shape of the design space can be precisely depicted and design curves can be well matched. Moreover, our CS-t achieves averaged 30x speedup compared to VPR placement. From the perspective of both design space estimation quality and runtime, our circuit similarity has been demonstrated to be a good tool for efficient FPGA design space exploration.

For future work, we will combine our circuit similarity with the existing pareto-point generation methodology [6]. In addition, we will try to apply our circuit similarity to other applications, *e.g.*, FPGA architecture design and FPGA verifications.

7. REFERENCES

[1] ABC: A System for Sequential Synthesis and Verification. `http://www.eecs.berkeley.edu/~alanmi/abc/`.

[2] SPIRAL: Software/Hardware Generation for DSP Algorithms. `http://spiral.ece.cmu.edu/mcm/gen.html`.

[3] A. G. Dempster and M. D. Macleod . Use of minimum-adder multiplier blocks in FIR digital filters. *IEEE Transactions in Circuits and Systems-II: Analog and Digital Signal Processing*, 42:569–577, 1995.

[4] A.M. Smith, J. Das, S.J.E. Wilton. Wirelength Modeling for Homogeneous and Heterogeneous FPGA Architectural Development. *ACM International Symposium on FPGAs*, 2009.

[5] H. Bunke. Error Correcting Graph Matching: On the Influence of the Underlying Cost Function. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21:917–922, 1999.

[6] D. Sheldon and F. Vahid. Making Good Points: Application-Specific Pareto-Point Generation for Design Space Exploration using Statistical Methods. *ACM International Symposium on FPGAs*, 2009.

[7] G. Jeh and J. Widom. A Measure of Structural-context Similarity. *Proceedings of the Eighth International Conference on Knowledge Discovery and Data Mining*, 2002.

[8] A. V. Goldberg. An Efficient Implementation of a Scaling Minimum-Cost Flow Algorithm. *Journal of Algorithms*, 22:1–29, 1997.

[9] J. Das, S.J.E. Wilton, W. Luk, P.H.W. Leong. Modeling Post-Techmapping and Post-Clustering FPGA Circuit Depth. *International Conference on Field-Programmable Logic*, 2009.

[10] M. L. Fernandez and G. Valiente. A Graph Distance Metric Combining Maximum Common Subgraph and Minimum Common Supergraph. *Pattern Recognition Letters*, 22:735–758, 2001.

[11] M. Rupp, E. Proschak and G. Schneider. Kernel Approach to Molecular Similarity Based on Iterative Graph Similarity. *Journal of Chemical Information and Modeling*, 2007.

[12] M. Xu and F. Kurdahi. Area and Timing Estimation for Lookup Table Based FPGAs. *European Design and Test Conference*, 1996.

[13] M. Pelillo. Matching Free Trees, Maximal Cliques and Monotone Game Dynamics. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24:1535–1541, 2002.

[14] R. Albert and A. L. Barabasi. Statistical Mechanics of Complex Networks. *Reviews of Modern Physics*, 74:47–97, 2002.

[15] S. Melnik, H. Garcia-Molina and A. Rahm. Similarity Flooding: A Versatile Graph Matching Algorithm and its Application to Schema Matching. *Proceedings of the 18th International Conference on Data Engineering*, 2002.

[16] T. Givargis and F. Vahid. Platune: A Tuning Framework for System-on-a-Chip Platforms. *IEEE Transactions on Computer Aided Design*, 21:1317–1327, 2002.

[17] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, USA, 2001.

[18] V. Betz and J. Rose. VPR: A New Packing, Placement and Routing Tool for FPGA Research. *International Workshop on Field Programmable Logic and Applications*, 1997.

[19] V. Blondel, A. Gajardo, M. Heymans, P. Senellart and P. Van Dooren. A Measure of Similarity between Graph Vertices: Applications to Synonym Extraction and Web Searching. *Society for Industrial and Applied Mathematics Review*, 46(4):647–666, 2004.

[20] Y. Voronenko and M. Pschel . Multiplierless Multiple Constant Multiplication. *ACM Transactions on Algorithms*, 2007.

[21] S. Yang. Logic Synthesis and Optimization Benchmarks User Guide, Version 3.0, 1991.

[22] L. Zager. *Graph Similarity and Matching*. PhD thesis, MIT, 2005.