Analyzing the Impact of Heterogeneous Blocks on FPGA Placement Quality

Chang Xu*, Wentai Zhang*, Guojie Luo*†
*Center for Energy-Efficient Computing and Applications (CECA),
School of EECS, Peking University, Beijing 100871, China
†PKU-UCLA Joint Research Institute in Science and Engineering
Email: changxu@pku.edu.cn, rchardx@gmail.com, gluo@pku.edu.cn

***Abstract*—In this paper we propose a quantitative approach to analyze the impact of heterogeneous blocks (H-blocks) on the FPGA placement quality. The basic idea is to construct synthetic heterogeneous placement benchmarks with known optimal wirelength to facilitate the quantitative analysis. To the best of our knowledge, this is the first work that enables the construction of wirelength-optimal heterogeneous placement examples. Besides analyzing the quality of existing placers, we further decompose the impacts of H-blocks from the architectural aspect and netlist aspect. Our analysis shows that a heterogeneous design hides the wirelength degradation by a more compact netlist than its homogeneous version; however, the heterogeneity results in a optimality gap of 52% in wirelength, where 25% is from architectural heterogeneity and 27% is from netlist heterogeneity. Therefore, new heterogeneous placement algorithms are needed to bridge the optimality gap and improve design quality.**

I. Introduction

Modern heterogeneous FPGA architecture contains not only configurable logic blocks (CLBs) but also heterogeneous blocks (H-blocks for short), such as: DSPs, BRAMs, etc. The introduction of H-blocks aims to improve performance and reduce area and power. Advanced FPGA synthesis algorithms are crucial for exploiting the heterogeneous resource efficiently, among which placement is an important stage to optimize the performance and power of on-chip interconnects.

The placement algorithm has been studied for decades. Basically, it can be divided into three classes: simulated annealing algorithm (e.g., VPR [1]), partition based algorithm (e.g., PPFF [2] and Michael et al. [3]), and analytical algorithm (e.g., HeAP [4] and Lin et al. [5]). More details of these algorithms can be found in [6] for FPGAs and [7] for ASICs. Despite the existence of well-developed placement algorithms, the appearance of H-blocks rise new problems. By far, existing placers only show their capability to handle H-blocks without discussing how their algorithms differ from the homogeneous case, such as VPR. Others do design different method with heterogeneity awareness. For example, HeAP adapts different rough legalization strategy for H-blocks. However, it is still unclear how well the placement algorithm handles the heterogeneity.

The heterogeneity mainly comes from two aspects: the architectural heterogeneity and the netlist heterogeneity. We observe that for the same homogeneous netlist without H-blocks, the wirelength quality of placement on homogeneous FPGAs differs from the quality of heterogeneous FPGAs. This impact is named *architectural heterogeneity* in this paper. Given the same heterogeneous FPGA, we can synthesize the RTL design into homogeneous netlist (without H-blocks) and heterogeneous netlist (with H-blocks), respectively. We observe that for a netlist with similar number of blocks, the wirelength quality of heterogeneous netlist differs from the quality of homogeneous netlists. This impact is named *netlist heterogeneity* in this paper. In the following analysis, we develop a systematic way to separately analyze the impact from architectural heterogeneity and netlist heterogeneity.

Our evaluation is based on the quantitative analysis: we first construct a synthetic netlist with *known optimal* wirelength. With such netlist, we further design experiments to quantify the impact from architectural heterogeneity and netlist heterogeneity respectively. We also evaluate the gap between existing placers, including VPR and Quartus, and the synthetic optimal placement. There are multiple previous works on synthetic benchmark generation. Chang et al. [8] firstly proposed the idea of generating placement benchmarks with known optimal wirelength (PEKO). However, the PEKO benchmarks are based on the assumption that all the blocks are of equal size and unique type, which is not suitable for heterogeneous FPGAs. Cong et al. [9] extended the PEKO idea to examine the optimality of mixed-size ASIC placers. However, H-blocks in FPGAs are different from macros in ASICs: macros in ASICs are freely movable, while H-blocks in FPGAs are only placeable in a limited set of tiles. Papa et al. [10] constructed benchmarks with structured optimal solutions that helped identify the issues of existing algorithms. Ward et al. [11] high-lighted the datapath placement problem, and constructed two customized designs that performed common logic functions with manual layouts. Again, all these existing placement benchmarks target ASIC placers but not FPGA placers.

In this paper, we make the following contributions:

- We quantify the impact of H-blocks on the wirelength quality of FPGA placement. Specifically, we can separately analyze such impact from two sources: architectural heterogeneity and netlist heterogeneity.
- To facilitate the analysis with synthetic hetereogeneous benchmarks with known optimal wirelength, we propose an effective *one-dimensional* search method to generate

978-1-4799-6245-7/14/$31.00 ©2014 IEEE

such heterogeneous netlists.

- We evaluate and analyze the heterogeneous placement quality, in terms of optimality gap, on popular academic and industrial FPGA placers.
- Our synthetic benchmarks for heterogeneous FPGA placement will be released in the public domain[1], available in VPR format, Altera VQM format [12], and probably in EDIF format with optimal placement information. Thus, researchers and developers of heteregeneous FPGA placers will be able to better understand the quality of their placers in handling the architectural heterogeneity and netlist heterogeneity separately.

The remainder of this paper is organized as follows. Section II qualitatively analyzes the complexity and inefficiency of the placement algorithm caused by the H-blocks. Before quantifying the impact of heterogeneity, Section III firstly introduces how to generate a synthetic netlist with known optimal wirelength with our one-dimensional search method. Section IV designs experiments to separately analyze the architectural heterogeneity and the netlist heterogeneity. Experimental study reported in Section V shows that there is still much room left for the improvement of heterogeneous FPGA placement. Finally, Section VI gives conclusions and future work.

II. Motivations

Based on our observations, the existence of H-blocks benefits the design by reducing the netlist size on one hand, and makes the placement more complicated on the other hand. Such observations motivate us to quantify the impact of H-blocks on the FPGA placement quality.

A. The Hidden Impact due to the Reduced Netlist Size

The H-blocks implement their own functionality in a more efficient way than CLBs, in terms of delay, area and power. By synthesizing such functionality into H-blocks, the whole design is benefit from performance improvement and area reduction.

For designs with identical functionality, mapping to a heterogeneous architecture will result in a more compact netlist. The statistics in Table II will show that both the number of logic blocks and the number of nets can reduce by about 30% compared with its homogeneous implementation. Details for such heterogeneous and homogeneous synthesis flows will be described in Section IV.

B. Intuitions of the Difficulties in Heterogeneous Placement

The heterogeneity that complicates the placement problem mainly comes from two aspects: architectural heterogeneity and netlist heterogeneity.

The architectural heterogeneity restricts the legal locations of blocks to their specific tiles. As shown in Figure 1, in the heterogeneous island-style FPGA, different logic resources are aligned in tiles on the discrete, fixed locations. For a legal placement, one type of blocks in the netlist can only be put in the tiles with a corresponding type; tiles with different types serve as obstacle during placement. Thus, the heterogeneous tiles make the placement much more complicated compared with the placement with homogeneous tiles only.

Given the heterogeneous architecture, a portion of logics in the design may map into H-blocks. Such netlist is a heterogeneous netlist, containing more than one type of blocks. The wirelength of heterogeneous net is affected by different types of blocks jointly, making the wirelength minimization more difficult. Such difficulty comes from netlist heterogeneity.

Take the simulated annealing-based (SA) placement algorithm as an example, we will discuss informally how architectural heterogeneity and netlist heterogeneity degrade placement quality.

Adopting Monte-Carlo-based sampling, SA optimizes wirelength by perturbing the locations of blocks with block movement or swapping. In order to avoid huge degradation on wirelength, block movement and swapping are restricted within a small region of $(2R_{limit}+1)\cdot(2R_{limit}+1)$ (shown in Figure 1), where R_{limit} is called the "range limit". The acceptance ratio of block movement is related with both temperature and movement strategy. According to Lam et.al, [13], good annealing schdule should decrease the temperature as fast as possible while maintaining the acceptance ratio at 0.44. This results in R_{limit} being the size of the entire chip for the first part of the annealing, shrinking gradually during the middle stages of the annealing, and finally being 1 logic block at low temperatures [14].

Because of the architectural heterogeneity, the existence of H-blocks reduces the number of candidate locations for block movement or swapping within the range limit. As shown in Figure 1, the logic block has fewer candidate locations when there have different type of blocks within the range limit. Such situation is even more serious as the range limit shrinks, affecting the acceptance rate at a low temperature. The limited exploration space will definitely affect the effectiveness of the simulated annealing.

On the other hand, the wirelength minimization for a heterogeneous net is more difficult than a homogeneous net. For example, in Figure 1 the movement distances for DSPs are larger than CLBs, resulting in higher probability of wirelength degradation, and thus lower probability of acceptance.

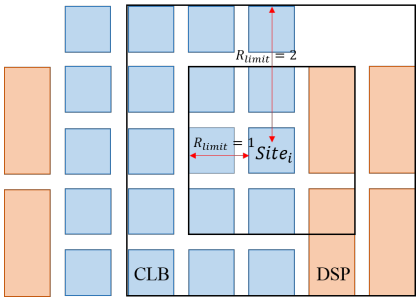

Fig. 1. Island-style heterogeneous FPGA and different range limits for block movement from $Site_i$.

[1]Please download the synthetic benchmarks from https://github.com/FPGAStudy/placement/

III. Synthetic Benchmark Generation

In Section II, we qualitatively analyze the issues that the heterogeity brings in. However, quantitative analysis is still required to determine the necessity of heterogeneity-aware placement algorithm. Before quantifying the optimality gap of wirelength, we will firstly describe how to generate our synthetic benchmarks with known optimal wirelength.

A. Basic Idea

Since the placement problem is NP-hard, it is impractical to produce an optimal placement with minimum wirelength for a realistic design in reasonable time. Instead of obtaining an optimal placement of a realistic design for analysis, we generate synthetic benchmarks that reserve some properties from realistic designs while providing known optimal wirelength by construction. Common netlist properties for reference include the number of nets and the number of type-wise pins in each net.

For example, given the extracted netlist properties from a realistic design (e.g., there are one homogeneous net $net1$ and one heterogeneous net $net2$ in Figure 2(a)) and the target architecture (in Figure 2(b)), we want to construct a netlist with known optimal wirelength while preserving these properties.

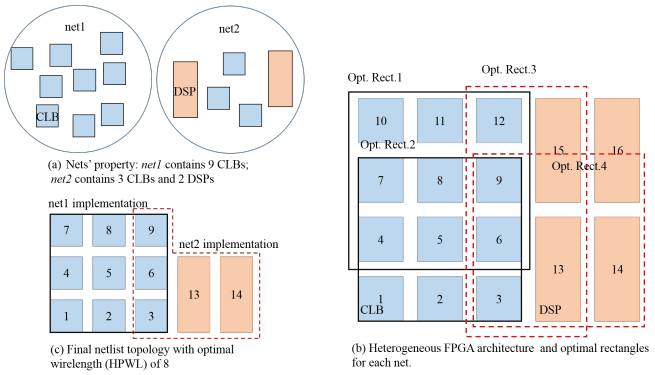

Fig. 2. Toy benchmark generation.

The netlist generation is done by implementing each net one by one. A net with optimal wirelength can be constructed by connecting blocks within one of its *optimal rectangles* (defined latter). In Section III-B, We propose the *one-dimensional search* method to obtain the optimal rectangles for a given heterogeneous net. Please note that in terms of wirelength, each net might have multiple optimal rectangles located at different tiles. These optimal rectangles are candidates for net implementation with the minimum half-perimeter wirelength (HPWL). For example in Figure 2(b), $Opt.Rect.1$ and $Opt.Rect.2$ are optimal rectangles for $net1$ with a minimum HPWL of 4, assuming the pins are located at the center of blocks; and $Opt.Rect.3$ and $Opt.Rect.4$ are optimal rectangles for $net2$ with a minimum HPWL of 4. Then the choice of net candidates is done with the strategy depicted in Section III-C. For example, one possible netlist implementation consisting of $net1$ and $net2$ is shown in Figure 2(c).

B. Generate Optimal Rectangles for a Heterogeneous Net

The optimal wirelength of a synthetic net is guaranteed by construction. In the homogeneous case, the wirelength lower bound for a net with degree p is quite easy to estimate [8]. If a net with p pins achieves the lower bound, all the pins are accommodated in their optimal rectangle with a size of:

$$\lceil\sqrt{p}\rceil \cdot \lceil\frac{p}{\lceil\sqrt{p}\rceil}\rceil \text{or} \lceil\frac{p}{\lceil\sqrt{p}\rceil}\rceil \cdot \lceil\sqrt{p}\rceil \quad (1)$$

The *optimal rectangle* for a net n is defined as a rectangle with minimum half perimeter that can accommodate all types of blocks in net n. Thus, the minimum HPWL of net n is calculated by the half-perimeter of the optimal rectangles enclosing its pins. Formally speaking, we represent an optimal rectangle with a 4-tuple (x, y, l_x, l_y), where (x, y) is the bottom-left corner (also called the *starting point* in the following) and l_x (or l_y) is the width (or height) of the optimal rectangle. Each net might have multiple optimal rectangles, which are the candidates for synthetic net implementation. The choice of candidates will be discussed in Section III-C.

However, for heterogeneous nets, different types of blocks have different geometric size and align in different tiles. There seems no simple expression like Equation 1 to determine the optimal rectangle for a heterogeneous net, since the optimal size is related with locations. With the example in Figure 3, rectangles with different starting points have different size to accommodate the logic blocks. The optimal rectangle must be the minimum one starting from arbitrary locations. Thus, it is difficult to determine the optimal size with a simple expression like Equation 1.

To solve this problem, we propose an effective one-dimensional search method to find the optimal rectangles for the heterogeneous net n. Firstly we solve a simpler problem by focusing on the rectangles with the same starting point s_i, and search among these rectangles for the ones with minimum perimeter to accommodate all types of blocks in the net n. And then we use the above solver to search among all possible tiles as the starting point to obtain the globally optimal rectangles. Please note that not all the tiles on the chip should be searched and compared. Only those *effective tiles* are necessary to be searched. The criteria of effective tile will be defined latter.

The basic idea of the *one-dimensional search* method is to enumerate the width or height of rectangle only instead of both dimensions (width and height). For example, if l_x is given, it is straightforward to compute the corresponding minimum l_y. After each enumeration of l_x, we can update the best-known smallest value of $l_x + l_y$, and keep incrementing l_x for the next enumeration until the new l_x is larger than the best-known smallest $l_x + l_y$. The search range for the l_x is called the *reasonable* l_x.

Take $net2$ in Figure 2 (a) as an example, we illustrate the process of one-dimensional search in Figure 3. Given the starting point s_i or s_j, we want to find the rectangle with minimum half perimeter to accommodate $net2$. Let l_x be the searching dimension, Table I shows the size of rectangles as

l_x increases gradually. We stop searching in the l_x dimension when l_x is larger than the best-known smallest l_x+l_y (7 when starting at S_i and 6 when starting at S_j). Compared with all the rectangles, the minimum half perimeter is 6 when taking s_j as the starting point. There are three optimal rectangles for $net2$, which are $(x_j, y_j, 2, 4)$, $(x_j, y_j, 3, 3)$, and $(x_j, y_j, 4, 2)$. The pseudo code of the one-dimensional search algorithm is also shown in Algorithm 1.

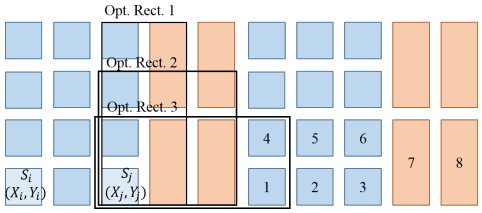

Fig. 3. Illustration to the one-dimensional search

TABLE I
RECTANGLES ACHIEVED IN THE SEARCH PROCESS BY TAKING $S_i(S_j)$ AS AS STARTING POINT

S_i						
l_x	1∼3	4	5	6	7	
l_y	∞	4	2	2	2	
l_x+l_y	∞	8	7	8	9	
S_j		$Rect.1$	$Rect.2$	$Rect.3$		
l_x	1	**2**	**3**	**4**	5	6
l_y	∞	**4**	**3**	**2**	2	2
l_x+l_y	∞	**6**	**6**	**6**	7	8

Since optimal rectanges have the minimum half-perimeter compared with other rectangles starting at arbitral tiles, each net implemented within its optimal rectangle must have minimum HPWL. However, in practice, we do not need to explore all the tiles over the FPGA chip, since modern FPGAs have column-based structures, and the columns usually have repeated patterns. We can find a minimum set of tiles that can represent all the tiles on chip. Such set of tiles is called the *effective tiles*. Obviously, exploring within the effective tiles can still guarantee the optimality. In practice, the number of effective tiles can be quite small. Take VPR architecture as an example, the heterogeneous architecture (defined in the XML file) will tell the starting point of the type-wise tiles and the intervals. Thus, the tiles within the intervals can represent the periodical appearance of other tiles. For example, the effective tiles for the architecture in the Figure 3 have only 2 DSPs (labeled 7,8) and 6 CLBs (labeled 1∼6). With the effective tiles, the searching space can be greatly reduced.

C. Generate a Heterogeneous Netlist with Known Optimal Wirelength

Netlist is generated with the flow in Figure 4. The net set N containing netlist properties is extracted from realistic designs. We construct each net N_i in set N until it is empty. The sequence of net implementation is determined by the number of optimal rectangles each net has. Basically, the heterogeneous net containing rare resources (e.g, DSPs, IOs) tends to have fewer candidates than the homogeneous net with only CLBs. A high-degree net is likely to have fewer optimal rectangles than a low-degree net. In our implementation, Net with fewer optimal rectangles will be generated first.

Algorithm 1 Find the optimal rectangles for a heterogeneous net

Require:
1) a heterogeneous FPGA
2) the number of type-wise blocks in the net e (labeled with $d(e,*)$)
Algorithm:
1: $HPWL_{OPT}$ = $+\infty$
2: OptRects := $\varnothing$
3: **for** each *effective* s_i **do**
4: (x_i, y_i) := the coordinate of s_i
5: **for** each *reasonable* l_x **do**
6: calculate the minimum l_y such that in the rectangle $(x_i, y_i) - (x_i + l_x, y_i + l_y)$, the number of blocks of type t is not less than $d(e,t)$
7: **if** $l_x + l_y < HPWL_{OPT}$ **then**
8: $HPWL_{OPT} := l_x + l_y$
9: OptRects:=(x_i, y_i, l_x, l_y)
10: **else if** $l_x + l_y == HPWL_{OPT}$ **then**
11: OptRects:=OptRects $\bigcup$ (x_i, y_i, l_x, l_y)
12: **end if**
13: **end for**
14: **end for**
15: **return** OptRects

With Algorithm 1, we can find the optimal rectangles for a given net N_i. In terms of net implementation candidate, we simply choose the bottom-left-most R_i first. If the input or output pins of the blocks in R_i are used up, we choose another optimal rectangle R_j. If the net cannot find an optimal rectangle with enough available pin resources, such net will be thrown away instead of finding a sub-optimal implementation. The netlist can be constructed when all the nets have been processed.

Please note that the final netlist properties might be different from the properties extracted from realistic designs. Since for each tile, the physical pin number is limited. What's more, if the heterogeneity is quite complex, the optimal wirelength can only obtained at a few number of tiles, making those tiles become hot spots. Thus, the latter arrived net will be difficult to find its accommodation because of the exhaust of pin resources. In this way, our benchmark generation is in a "best-effort" style.

D. Time Complexity Analysis

A pessimistic estimation for the number of *reasonable* l_x to be searched at one starting tile is kT, where k is the net degree and T is the number of block types in the net. The optimal rectangles can be achieved by searching tiles in the *effective* tiles. Supposing the *effective* tiles lie within the range of T_xT_y.

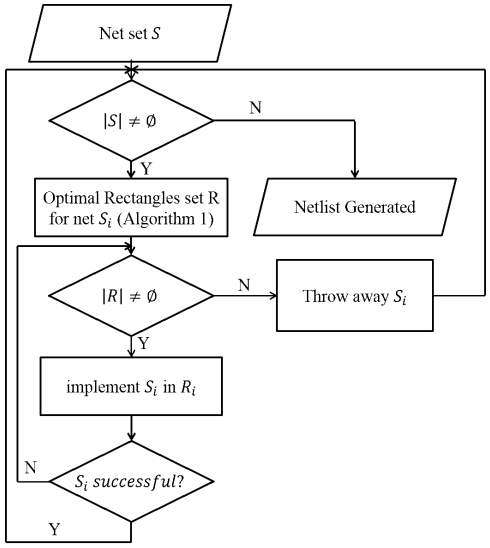

Fig. 4. Netlist generation flow

The searching time spends on one net equals to $T_x T_y kT$. The overall pessimistic estimation of the runtime is $O(T_x T_y kTN)$, where N is the number of nets.

In practice, the runtime can be much smaller than this estimation, and it is possible to speed up the benchmark generation by building a lookup table of optimal rectangles for a specific FPGA architecture.

IV. Experimental Flow for the Quantitative Analysis

In this section we describe the evaluation flow to quantify wirelength gaps of state-of-the-art placers, including VPR and Quartus. And we further design experiments to separately analyze the impacts of architectural heterogeneity and netlist heterogeneity.

A. Evaluation Flow

There are multiple choices for FPGA synthesis: 1) synthesize the RTL design into a homogeneous netlist and map on a homogeneous FPGA; 2) synthesize the RTL design into a homogeneous netlist and map on a heterogeneous FPGA; 3) synthesize the RTL design into a heterogeneous netlist and map on a heterogeneous FPGA. In this section, we depict the evaluation flow to calculate wirelength gaps of these three different choices on the academic placer VPR[2] and the industrial placer Quartus[3].

As shown in Figure 5, the evaluation flow takes an RTL design as input and output the wirelength gap of VPR placer or Quartus placer. The whole flow consists of three steps: 1) synthesize the RTL design into a homogeneous or heterogeneous netlist (as a reference netlist) with front-end synthesis flow; 2) given the reference netlist and the target architecture, we use the Algorithm 1 in Section III to generate a synthetic benchmark with known optimal wirelength; 3) feed VPR placer and Quartus placer with the synthetic benchmark and obtain the wirelength gap towards optimal placement solution.

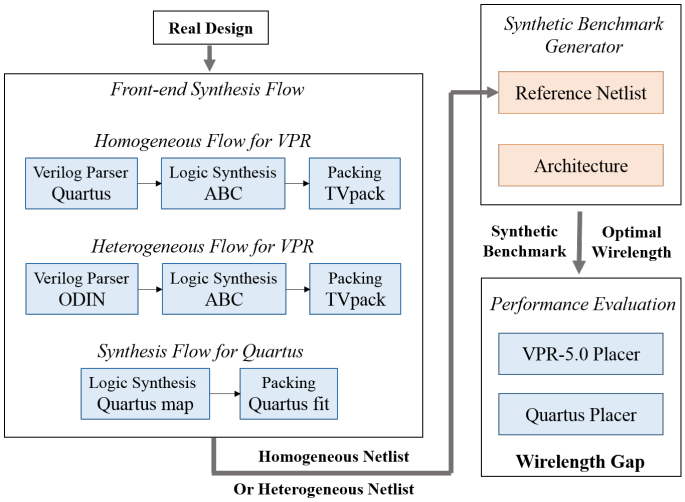

Fig. 5. The synthesis flow: (a) homogeneous flow (b) heterogeneous flow

We have customized the setting files of synthesis tools to eliminate the nosing impacts from these following parts:

- Different optimization efforts of front-end synthesis tools: ODIN and Quartus. We use ODIN and Quartus to parse RTL designs into BLIF format targeting heterogeneous VPR architecture and homogeneous VPR architecture respectively. The BLIF files are generated without optimization, so that ODIN and Quartus in the front end only serve as RTL parsers. For example, we turn on the option of "dump_blif_before_optimization" in Quartus verilog parsing. The optimization of BLIF netlist is then left to ABC.
- Placers pay much efforts to optimize critical paths. For both VPR placer and Quartus placer, we specify the "wirelength-driven" placement mode. For example, in VPR, we set the placement to be "bounding-box driven placement". And in Quartus, we turn off the timing optimization with setting "optimize timing off".
- Quartus placer does not only placement but also packing. The synthetic netlist (in VQM format) generated is the LAB-level netlist. In order to prevent the packer from disturbing the packing at the BLE-level, we use the feature of "virtual pin" and "logic lock" to lock the BLE blocks within the same "LAB".

Based on different netlist type and architecture type, our evaluation flow can be classified into three flows:

- Flow-A: generate a *homogeneous* netlist with known optimal wirelength targeting a *homogeneous* architecture.
- Flow-B: generate a *homogeneous* netlist with known optimal wirelength targeting a *heterogeneous* architecture.

[2]We use VPR-5.0 placer because the placement algorithms in VPR-5.0 and VPR-7.0 do not have significant changes. The architecture we used is defined in "K4-n12.xml".

[3]We use Quartus 12.0 and Cyclone II FPGA device.

- Flow-C: generate a *heterogeneous* netlist with known optimal wirelength targeting a *heterogeneous* architecture.

Please note that the netlist topology may be different for *Flow-A* and *Flow-B* even with the same reference netlist, since the optimal rectangles are different for different architectures. These three flows will be used to evaluate the placement quality of VPR placer and Quartus placer in Section V-A.

B. Separate the Impacts of Architectural and Netlist Heterogeneity

For the sake of quantifying the impacts of architectural and netlist heterogeneity, a direct comparison of performance using the synthetic benchmarks generated with *Flow-A*, *Flow-B*, and *Flow-C* cannot work due to the difference of netlist topology. In order to remove the impact of netlist topology, we design three cases to separate the impact of architecture's heterogeneity from the netlist heterogeneity.

- Case-1: Given the reference netlist, we will use *Flow-A* to generate the synthetic netlist with known optimal wirelength targeting the homogeneous architecture.
- Case-2: By adding some heterogeneous tiles, we expand the homogeneous architecture into a heterogeneous architecture, while keeping the netlist unchanged. With some simple computation, we can get an upper-bound of the optimal wirelength for Case-2.
- Case-3: Based on Case-2, some H-blocks blocks will be added into the homogeneous netlist to generate the heterogeneous netlist. Similar to Case-2, we can also get a upper-bound of the optimal wirelength for Case-3.

By comparing the placement results in *Case-1* and *Case-2*, we can get the amount of impact from architecture heterogeneity. While comparing the placement results of *Case-2* and *Case-3*, we can quantify the impact from netlist heterogeneity on wirelength.

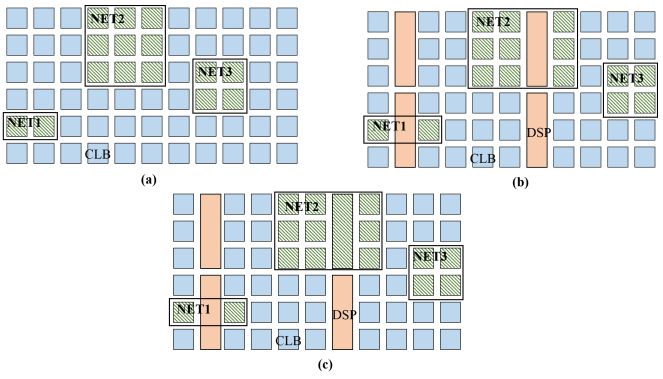

Fig. 6. (a) Case-1: placing homogeneous netlist on the homogeneous architecture. (b) Case-2: transforming the homogeneous architecture to the heterogeneous architecture. (c) Case-3: transforming the homogeneous netlist into the heterogeneous netlist.

Using the example depicted in the Figure 6, we further explain the details. In the Figure 6(a), we have generated the homogeneous netlist with known optimal wirelength with *Flow-A*, which containing a 2-degree net ($net1$), a 9-degree net ($net2$), and a 4-degree net ($net3$). The total optimal wirelength is 7.

Then the heterogeneous tiles (e.g, DSPs) will be inserted into the homogeneous architecture, as shown in Figure 6 (b). The number of tiles to be added and where to insert those tiles are directed by the real heterogeneous FPGA architecture. For example, in the heterogeneous VPR architecture, the starting point of the heterogeneous tiles and the interval for periodical appearance are defined in the architecture file. After transforming the homogeneous architecture into the heterogeneous architecture, we will calculate the wirelength for the stretched placement after heterogeneous tile insertion, which is an upper-bound of the optimal wirelength. As shown in Figure 6(b), nets spanning the heterogeneous sites are no longer optimal (i.e. $net1$ and $net2$ are no longer optimal). So 9 is a upper-bound of the optimal wirelength.

Then we will add some H-blocks into the netlist to transform it into a heterogeneous netlist. Please note that we only add H-blocks for those nets, whose wirelength will not be increased after adding these H-blocks. For example, in Figure 6(c) we only add H-blocks for $net2$. Thus, with such constraint, the upper-bound for the optimal wirelength in Figure 6(c) is same with wirelength for Figure 6(b). The wirelength upper-bound is still 9.

With the netlists generated according to these three cases, we can perform comparisons to separately quantify the impacts of architectural heterogeneity and netlist heterogeneity. The statistics will be shown in Section V-B.

V. Experimental Results and Discussions

Our synthetic benchmark generator is implemented in C++ and compiled with G++ 4.5.1. The generator runs on Intel Xeon CPU E5620 Linux work station. Realistic RTL designs for reference are either from the VPR benchmark suite or from the QUIP benchmark suite. In order to test the scalability, we also implement two large designs (fft and ifft) in verilog.

With the flow shown in Figure 5, we synthesize the realistic RTL designs into the reference netlists. Properties of reference netlists are depicted in Table II, including the number of nets (#Net) and the number of blocks (#CLB, #DSP, #LAB).

The synthetic netlists will reserve the features of these realistic benchmarks at the best effort. By comparing the number of net (Diff1) and the number of CLB (Diff2) of homogeneous netlist and heterogeneous netlist, we can see that for the same design, synthesizing into a heterogeneous netlist can save around 30% resource utilization compared to a homogeneous netlist.

The name of synthetic netlist generated by referring the specific realistic design is listed in the column of "Syn.name" in Table II, which will be used in the following experiments. Please note that both Bench.aX and Bench.bX are homogeneous netlists generated from the same realistic designs. However, they are targeting different architectures, homogeneous architecture and heterogeneous architecture respectively. Thus, both the optimal wirelength and netlist topology are different.

TABLE II
PROPERTIES OF REFERENCE NETLISTS FOR VPR

Reference Benchmark for VPR	*Homogeneous Netlist(10)*			*Heterogeneous Netlist(8)*				*Diff1*	*Diff2*
	#Net	*#CLB*	*Syn.name*	*#Net*	*#CLB*	*#DSP*	*Syn.name*	*(#net)*	*(#CLB)*
iir1	744	109	Bench.a1(b1)	487	70	5	Bench.c1	35%	31%
fir_scu_rtl	777	117	Bench.a2(b2)	544	70	17	Bench.c2	30%	26%
diffeq_f_systemC	2230	310	Bench.a3(b3)	1596	207	4	Bench.c3	28%	32%
oc54_cpu	2182	362	Bench.a4(b4)	1777	301	1	Bench.c4	19%	17%
stereovision1	18866	2909	Bench.a5(b5)	12100	1583	152	Bench.c5	36%	46%
diffeq_paj_convert	2362	327	Bench.a6(b6)						
cf_cordic_v_18	3493	561	Bench.a7(b7)						
des_perf	4461	849	Bench.a8(b8)						
fft	68841	10704	Bench.a9(b9)						
ifft	75873	10712	Bench.a10(b10)						
cf_fir_24				3175	366	25	Bench.c6		
stereovision2				32095	3938	564	Bench.c7		
paj_top_hier				40231	6075	6	Bench.c8		

TABLE III
WIRELENGTH GAP OF VPR PLACER WITH *Flowa*, *Flowb* AND *Flowc*

Hom.Netlist+Hom.Arch. (Flow-A)				*Hom.Netlist+Het.Arch. (Flow-B)*				*Het.Netlist+Het.Arch. (Flow-C)*			
Bench.	OPT_{wl}	VPR_{wl}	*WG*	*Bench.*	OPT_{wl}	VPR_{wl}	*WG*	*Bench.*	OPT_{wl}	VPR_{wl}	*WG*
Bench.a1	1158	1315	14%	*Bench.b1*	1103	1522	38%	*Bench.c1*	837	917	9%
Bench.a2	1172	1514	29%	*Bench.b2*	1197	1592	32%	*Bench.c2*	988	1299	31%
Bench.a3	3014	3824	27%	*Bench.b3*	3117	4614	47%	*Bench.c3*	2196	2945	34%
Bench.a4	3390	4392	30%	*Bench.b4*	3475	4999	43%	*Bench.c4*	2868	4199	46%
Bench.a5	26426	36086	36%	*Bench.b5*	27684	38447	37%	*Bench.c5*	15557	25899	66%
Bench.a6	3007	3517	17%	*Bench.b6*	3246	4100	25%	*Bench.c6*	4400	6715	53%
Bench.a7	5136	6461	26%	*Bench.b7*	5321	7280	36%	*Bench.c7*	50996	98576	93%
Bench.a8	7136	9400	32%	*Bench.b8*	7011	9448	34%	*Bench.c8*	62185	95860	54%
Bench.a9	88369	129705	47%	*Bench.b9*	109412	167498	52%	**Avg.**	-	-	48%
Bench.a10	88458	155483	76%	*Bench.b10*	108975	171020	56%				
Avg.	-	-	33%	**Avg.**	-	-	40%				

A. Evaluating the Quality of Existing Placers

With evaluation flows (*Flow-A*, *Flow-B*, and *Flow-C*) in Section IV-A, we have quantified the quality of VPR placer and Quartus placer on wirelength. The wirelength gap (WG) is defined in Equation 2, where $WL_{realistic}$ is the HPWL after VPR placement or Quartus placement and $WL_{optimal/upperbound}$ is the optimal or upper-bound wirelength for our synthetic benchmarks.

$$WG = \frac{WL_{realistic} - WL_{optimal/upperbound}}{WL_{optimal/upperbound}} \quad (2)$$

The placement quality of VPR is evaluated with synthetic benchmarks generated by *Flow-A*, *Flow-B*, and *Flow-C* respectively. WG for each condition is shown in Table III, where OPT_{wl} is the optimal wirlength and VPR_{wl} is the wirelength of VPR placer. Statistics show that:

- When placing homogeneous netlist on homogeneous architecture, WG is around **33%**.
- When placing homogeneous netlist on heterogeneous architecture, WG is around **40%**.
- When placing heterogeneous netlist on heterogeneous architecture, WG is **48%** on average.
- The reduction around **30%** problem size(in Table II) almost shows no advantage when compare the performance of *Flow-B* and *Flow-C*. The benefit of the problem size reduction is canceled out by the wirelength degradation caused by heterogeneity.

Thus, we call for efficient and effective heterogeneity-aware placer to eliminate the degradation of H-blocks.

For the Quartus placer, we have generated heterogeneous netlist with *Flow-C* targeting the Cyclone II FPGA device. The notation of OPT_{wl}, $Quartus_{wl}$, and WG are optimal wirelength, wirelength reported by Quartus, and the wirelength gap respectively. Statistics in the Table IV show that there are around **116%** wirelength gap on average.

TABLE IV
PROPERTIES OF REFERENCE NETLIST FOR QUARTUS AND WIRELENGTH GAP WITH *Flowc*

Reference Netlist Property				*Het.Netlist+Het.Arch.(Flow-C)*		
Design	*#Net*	*#LAB*	*#DSP*	OPT_{wl}	$Quartus_{wl}$	*WG*
nut_004	111	29	1	267	493	85%
nut_001	502	130	4	1193	2561	115%
fip_risc8	826	218	0	1852	4618	149%
mux8_128bit	624	112	0	1006	2051	104%
os_blofish	1533	463	0	3360	5041	50%
paj_rav	1620	387	18	3148	5227	66%
cfCordic_v18	1902	429	0	3117	8213	163%
oc54_cpu	1437	432	3	3609	9011	150%
mac1	1352	411	0	3612	8128	125%
des_perf	2270	1006	0	6948	17598	153%
Avg.						116%

B. Quantify the Impacts from Heterogeneity

In order to further separate the impacts between architectural heterogeneity and netlist heterogeneity, we have generated the synthetic netlists corresponding to the *Case-1*, *Case-2*, and *Case-3* in Section IV-B. For each synthetic netlist, we will give either the optimal wirelength or an upper bound. These synthetic netlists will feed in the VPR placer to get the placement wirelength.

The WG for three cases have been shown in Figure 7. Results show that:

- Architectural heterogeneity alone introduces at least an extra 25% of WG.
- Netlist heterogeneity by adding only a small number of DSPs (5.2% of the total number of blocks on average) can introduce additional 27% WG.

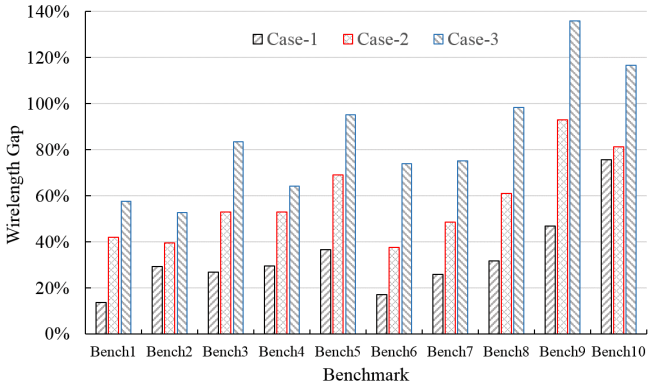

Fig. 7. Quantify the heterogeneity's impact on wirelength

VI. Conclusion and Future Work

To summarize, we construct synthetic benchmarks with known optimal placement, in terms of wirelength, with the presence of H-blocks. Specifically, we propose an efficient one-dimensional search method to generate a heterogeneous net with known optimal wirelength.

The benchmarking results of the academic FPGA placer VPR show that: (i) the average optimality gap is 9% to 93% for heterogeneous placement, compared with the average gap of 14% to 76% for homogenous placement; (ii) the degradation of wirelength by H-blocks cancel out the benefit of problem size reduction; (iii) the heterogeneous architecture alone degrades the placement quality by over 25%; (iv) the heterogeneous netlist degrades the quality by another 27%.

One potential problem of such analysis is that the benchmarks are synthetic may not represent the characteristics of the real-world designs. As a future work, we are going to explore possible metrics that can measure the "difficulty" of placement for a given netlist. If the synthetic benchmarks are easier to place than real-world designs, we have more confidence the gap analysis results are informative, because the gap that we obtain is smaller than the actual optimality gap. Otherwise, we have to resort to other methods to generate benchmarks with known optimal wirelength that are easier to place.

Acknowledgment

The author would like to thank Prof. Tao Wang and Jiahua Chen for providing the fft and ifft verilog code. The author would also like to thank Prof. Jason Cong, Prof. Kenneth Kent and Prof. Jonathan Rose for help with using Quartus and ODIN.

This work is partly supported by National Natural Science Foundation of China (NSFC) Grant 61202073, Research Fund for the Doctoral Program of Higher Education of China (MoE/RFDP) Grant 20120001120124, and Beijing Natural Science Foundation (BJNSF) Grant 4142022.

References

[1] V. Betz and J. Rose, "Vpr: A new packing, placement and routing tool for fpga research," in *Proceedings of the 7th International Workshop on Field-Programmable Logic and Applications*, ser. FPL '97. London, UK, UK: Springer-Verlag, 1997, pp. 213–222. [Online]. Available: http://dl.acm.org/citation.cfm?id=647924.738755

[2] P. Maidee, C. Ababei, and K. Bazargan, "Timing-driven partitioning-based placement for island style fpgas," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 24, no. 3, pp. 395–406, March 2005.

[3] M. J. Alexander, J. P. Cohoon, J. L. Ganley, and G. Robins, "Placement and routing for performance-oriented fpga layout."

[4] M. Gort and J. Anderson, "Analytical placement for heterogeneous fpgas," in *Field Programmable Logic and Applications (FPL), 2012 22nd International Conference on*, Aug 2012, pp. 143–150.

[5] T.-H. Lin, P. Banerjee, and Y.-W. Chang, "An efficient and effective analytical placer for fpgas," in *Proceedings of the 50th Annual Design Automation Conference*, ser. DAC '13. New York, NY, USA: ACM, 2013, pp. 10:1–10:6. [Online]. Available: http://doi.acm.org/10.1145/2463209.2488746

[6] D. Chen, J. Cong, and P. Pan, "Fpga design automation: A survey," *Found. Trends Electron. Des. Autom.*, vol. 1, no. 3, pp. 139–169, Jan. 2006. [Online]. Available: http://dx.doi.org/10.1561/1000000003

[7] G.-J. Nam and J. Cong, *Modern Circuit Placement: Best Practices and Results*, 1st ed. Springer Publishing Company, Incorporated, 2007.

[8] C.-C. Chang, J. Cong, M. Romesis, and M. Xie, "Optimality and scalability study of existing placement algorithms," *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, vol. 23, no. 4, pp. 537–549, Nov. 2006. [Online]. Available: http://dx.doi.org/10.1109/TCAD.2004.825870

[9] J. Cong, M. Romesis, J. Shinnerl, K. Sze, and M. Xie, "Locality and utilization in placement suboptimality," in *Modern Circuit Placement*, ser. Series on Integrated Circuits and Systems, G.-J. Nam and J. Cong, Eds. Springer US, 2007, pp. 13–36.

[10] D. A. Papa, S. N. Adya, and I. L. Markov, "Constructive benchmarking for placement," in *Proceedings of the 14th ACM Great Lakes Symposium on VLSI*, ser. GLSVLSI '04. New York, NY, USA: ACM, 2004, pp. 113–118. [Online]. Available: http://doi.acm.org/10.1145/988952.988981

[11] S. I. Ward, D. A. Papa, Z. Li, C. N. Sze, C. J. Alpert, and E. Swartzlander, "Quantifying academic placer performance on custom designs," in *Proceedings of the 2011 International Symposium on Physical Design*, ser. ISPD '11. New York, NY, USA: ACM, 2011, pp. 91–98. [Online]. Available: http://doi.acm.org/10.1145/1960397.1960420

[12] A. Cooperation, "Quartus ii handbook version 12.1," in *Quartus II Handbook*, 2012.

[13] J. Lam and J.-M. Delosme, "Performance of a new annealing schedule," in *Proceedings of the 25th ACM/IEEE Design Automation Conference*, ser. DAC '88. Los Alamitos, CA, USA: IEEE Computer Society Press, 1988, pp. 306–311. [Online]. Available: http://dl.acm.org/citation.cfm?id=285730.285780

[14] V. Betz, J. Rose, and A. Marquardt, Eds., *Architecture and CAD for Deep-Submicron FPGAs*. Norwell, MA, USA: Kluwer Academic Publishers, 1999.