How Much Does Regularity Help FPGA Placement?

Hongxin Kong[1], Lang Feng[1,3], Chunhua Deng[2], Bo Yuan[2], and Jiang Hu[1]

[1]Department of Electrical & Computer Engineering, Texas A&M University
[2]Department of Electrical & Computer Engineering, Rutgers University
[3]School of Electronic Science and Engineering, Nanjing University

***Abstract*—Placement plays a key role in determining FPGA circuit characteristics. Meanwhile, its long computation time is an important factor that makes the flexibility of FPGA computing much less competitive than software compiling. One observation is that there is an increasing need for designs with regularity. A particular example is systolic array-based neural network circuit design. In this work, FPGA placement exploiting design regularity is studied. For neural network designs with systolic arrays, our proposed regularity-aware approach achieves 2X to 28X speed up versus Versatile Place and Route (VPR) with limited circuit performance loss. At the same time, its solution quality is almost perfectly correlated with VPR and thus renders its role for early prototyping.**

I. Introduction

Field Programmable Gate Arrays (FPGA) is an increasingly popular computing platform due to its unique tradeoff between computing efficiency and application flexibility. In general, FPGAs are more efficient than software computing on general purpose microprocessors as the overhead associated with instructions is avoided. Compared to Application-Specific Integrated Circuit (ASIC) circuits that are fixed to specific applications, FPGA circuits provide better flexibility because of their reconfigurability. One example of large-scale industrial adoption is FPGA-based accelerators at Microsoft datacenters [1]. However, FPGA circuit design time is still nowhere near software programming. There are two primary reasons. First, FPGA design is rarely a single-shot process although automatic synthesis is available. There are numerous options for high level decisions, such as loop unrolling factor and number of pipelining stages, whose effects need to be carefully assessed. Actually, this is a main subject for design space exploration [2]. Second, the effect of a high level decision depends on subsequent physical design and thus layout aware high level synthesis is motivated [3]. However, physical design is perhaps the most time consuming part among all FPGA design steps. For instance, placement of a Convolutional Neural Network (CNN) circuit by Versatile Place and Route (VPR) [4], [5], which is the dominating academic tool, may take over 10 hours. High level models [6] can help prune out some inferior solutions. Nonetheless, several trials of physical design are still needed to achieve design closure. Although circuit performance is typically a primary design goal, the large variety of FPGA users includes those who accept mediocre designs in exchange for fast deployment [7].

In this work, we investigate fast FPGA placement for designs with regularity. General FPGA placement has been studied for many years [4], [5], [8]–[17] while there has been little attention [18] to designs with regularity. The regularities of the few existing works are fine-grained [18] or at chip die level [19], [20], which are easily obtained in a copy-and-paste manner. We consider middle-grained and relative general cases, where interactions among intra-PE (Processing Element), inter-PE and non-PE connections are simultaneously addressed.

Is it worthwhile to develop FPGA placement tools dedicated to designs with regularity? The answer depends on answers to two sub-questions: (1) is the practical need large enough? (2) is the benefit from regularity large enough? The wide applications of designs with regularity [21]–[28] indicate an arguably positive answer to the first sub-question. For instance, in FPGA computing, a rapidly growing application is neural network, which is often implemented in a very regular way such as systolic array [21], [26]–[28]. The structural regularity of systolic arrays makes it natural for systolic array placement to also have spatial regularity. Moreover, regularity can also facilitates routability. When local routability in one PE of an array is achieved, which is relatively easy, the global routability of an entire array is also ensured due to the spatial regularity. More broadly, systolic array-based FPGA designs have been widely used in many other important applications such as matrix multiplication [23], image processing [22] and digital signal processing [24]. As such, an important goal of this work is to find an answer to the second sub-question.

In the past, FPGA placement techniques were used to be dominated by simulated annealing [4], [5], [8], [10], [13] with VPR [4], [5] being its famous representative. The recent trend is analytical placement [9], [11], [12], [14]–[16] due to its capability of handling large designs. Nonetheless, simulated annealing and other probabilistic meta-heuristics [20] are still valuable due to their flexibility and easiness of parallelization [10], [13].

Therefore, they are still being used in recent research [17], [20] and serve as a key component in a commercial FPGA placement tool [29].

The regularity-aware placement in this work is a hybrid approach of Integer Linear Programming (ILP) and simulated annealing. In general, ILP is difficult to scale for large problems. However, regularity allows a significant reduction of decision variables and the ILP here is applied to only the regular portion of a design. In fact, the complexity of the ILP is independent of the number of Processing Elements (PEs) in a given PE array. The regularity allows speedup techniques for the simulated annealing part as well. The benefit of fast computation speed is mostly embodied for large designs, where computing acceleration is the most needed. For neural network designs with 64×64 systolic arrays, the proposed approach achieves over $25\times$ speedup compared to

VPR. The expense is no more than 10% degradation on post-routing wirelength and circuit frequency. Partly due to the regularity, the placement solutions are generally routable and routing completion is easily obtained for million-LUT designs. At the same time, the regular placement solution quality is almost perfectly correlated with VPR solutions. These results imply two possible application scenarios for the regularity-aware placement.

1) Applications where time to market is very tight while design specifications are not aggressive. Then, the regular placement can be directly adopted in final products.
2) Fast prototyping [30] for assessing the impact of various "what-if" options is achieved by the regularity-aware placement while the final placement solution is obtained from another tool.

The contributions of this work are summarized as follows.

- To the best of our knowledge, this is the first *algorithmic* study (instead of copy-and-paste implementation) of FPGA placement for systolic array considering regularity, which is a design style with increasing popularity.
- The study result shows that regularity allows over $25\times$ speedup for large neural network designs.
- An ILP approach to PE array placement is developed, including techniques for transforming nonlinear constraints to linear ones.
- Multiple regularity-enabled techniques are proposed for accelerating the simulated annealing in FPGA placement.
- The price paid for regularity is assessed and found to be no more than 10% degradation on wirelength and frequency.
- A nearly perfect correlation is identified between the solution quality of regularity-aware placement and that of a classical approach.

II. Previous Work

A classical approach to FPGA placement is simulated annealing, which is represented by the famous academic tool VPR [4], [5]. It is a very flexible framework that works for almost any objective function and does not rely on special problem structure. Its main drawback is the slow convergence for large problems. Therefore, later simulated annealing-based FPGA placement methods [8], [10], [13], [17] mostly focus on its computing acceleration. In particular, $10\times$ speedup is achieved by GPU-based parallelism [10] with no more than 3% increase of wirelength. Placement runtime and solution tradeoff was studied in [7], where the solution degradation can be as much as 33%.

Recently, analytical approach has been gaining popularity for FPGA placement [9], [11], [12], [14]–[16] due to its capability of handling large designs. Packing, which is used to be a standalone step prior to placement [5], [15], is integrated with placement in [16] to improve solution quality. Placement objectives also shift from minimizing wirelength [5], [9] to additionally considering routability [14]–[16].

There are very few works on FPGA placement considering regularity. One example [18] is for regular datapath placement. However, the regular arrays here are at the granularity of Configurable Logic Blocks (CLBs) and thus the corresponding placement is trivially laying out in a CLB array. By contrast, a PE in a neural network usually covers multiple CLBs and such PE array placement becomes non-trivial at all. Systolic array placement on FPGA is investigated in [20]. However, its main focus is on the utilization of DSP and RAM blocks instead of regularity. High level design reuse for modular designs is introduced in RapidWright [19] and is a complement to our algorithm level regularity. For example, the work of [20] achieves reuse and replication of SLRs (Super Logic Region) through RapidWright [19]. An SLR is a complete chip die, which is a much coarser grained than our PE-level regularity, where a PE covers around 10 CLBs. While the regularity in [19], [20] is realized in a copy-and-paste manner, our middle-grained regularity is more complicated to handle. Recently, FPGA routing considering regularity is studied in [31].

III. Regularity-Aware FPGA Placement

A. Overview of the Proposed Methodology

The input is a packed netlist corresponding to CLBs, Random Access Memory (RAM) blocks, Digital Signal Processors (DSPs) and IO pads, which need to be mapped to a given FPGA architecture. We consider circuit designs that contain a regular PE array besides non-regular elements or control logic. In general, a PE consists of multiple CLBs and this is particularly true for neural network circuits. The objective is to minimize total wirelength while all elements are placed legally. In addition, circuits in two different PEs are placed in the same way, i.e., the placement solutions of two PEs are identical. Last but not the least, PEs are placed in a geometrically regular array.

The regularity-aware placement includes two phases:

- Phase I: ILP (Integer Linear Programming)-based PE array placement.
- Phase II: Accelerated simulated annealing taking the PE array placement as a part of the initial solution.

In general, there are four types of wire connections to be considered.

1) Intra-PE nets. These are nets connecting blocks within a PE.
2) Inter-PE nets. These are nets involving connections among different PEs.
3) PE-external nets. These are nets involving connections between a PE and elements outside of the PE array.
4) External nets. These are nets that are completely outside of any PEs.

Phase I is focused on minimizing wirelength of intra-PE and inter-PE nets while phase II considers all types of nets.

B. Phase I: ILP-based PE Array Placement

The input of phase I is an $m \times n$ array of PEs $\mathcal{P} = \{\pi^{1,1}, \pi^{1,2}, ..., \pi^{1,n}, \pi^{2,1}, ..., \pi^{m,n}\}$. Each PE $\pi^{i,j} \in \mathcal{P}$ consists of k blocks $\pi^{i,j} = \{b_1^{i,j}, b_2^{i,j}, ..., b_k^{i,j}\}$. Figure 1 shows an example for $m = n = k = 3$. In phase I, the $m \times n$ array

structure is retained as layout pattern as well. The placement here is mostly to decide block locations within a PE and the same block placement pattern is shared by all PEs. For example, in Figure 1, the placement of the 3 blocks in a PE is shared by all the 9 PEs. The placement of a block $b_l^{i,j}$ is indicated by two decision variables, column index $x_l^{i,j}$ and row index $y_l^{i,j}$ in a given FPGA architecture. Due to the regularity, the overall number of decision variables is reduced from $2m \cdot n \cdot k$ to the order of $2k$. The large reduction makes ILP, which is computationally expensive, a practical component in the proposed methodology. For neural network circuits, the value of k is typically around a dozen.

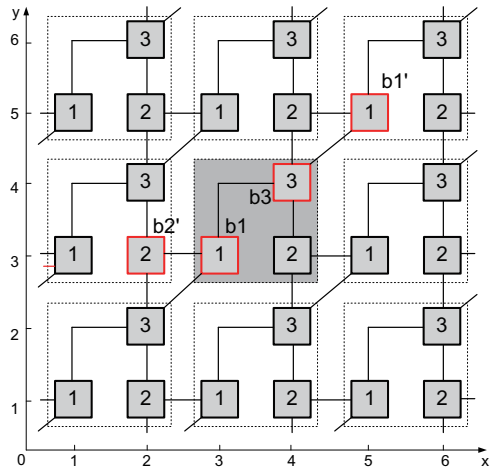


Fig. 1: An example of PE array placement. Each dotted box is a PE and and the shaded one is the reference PE. Each small square is a block.

Since virtually only the placement of one PE needs to be decided, one PE $\pi_{ref} = \{b_1, b_2, ..., b_k\}$ that is not adjacent to the array boundary is designated as the reference PE for this mission. For the sake of brevity, the PE row/column indices are skipped for the placement variables of the reference PE and its block locations are denoted as $\mathbf{x} = (x_1, x_2, ..., x_k)$ and $\mathbf{y} = (y_1, y_2, ..., y_k)$. For example, the location of block b_3 in Figure 1 is at $(x_3 = 4, y_3 = 4)$. Note that these decision variables must be integers in a legitimate range. The locations of the other PEs repeat the pattern of $(\mathbf{x}, \mathbf{y})$ with horizontal and/or vertical shift in terms of PE width and height given by:

$$W = \max_{l=1}^{k} x_l - \min_{l=1}^{k} x_l + 1, \quad H = \max_{l=1}^{k} y_l - \min_{l=1}^{k} y_l + 1$$

In Figure 1, $W = H = 2$. Please note the placement solution of phase I is an approximation as a uniform array is assumed for FPGA while an actual FPGA architecture is a mixture of CLBs, DSPs and RAM blocks.

The objective is to minimize Half Perimeter Wire Length (HPWL) for all intra-PE and inter-PE nets, e.g., intra-PE net (b_1, b_3), inter-PE nets (b_1', b_3) and (b_1, b_2') in Figure 1. The set of intra-PE nets $\mathcal{N}_{in}$ consists all nets that are completely within the reference PE. Each net in the set of inter-PE nets $\mathcal{N}_{\xi}$ spans at least one block in the reference PE and at least another block in a non-reference PE. We use a generic notation $\{b_1', b_2', ..., b_k'\}$ for indicating blocks in non-reference PEs, and vector pair $\mathbf{x}' = (x_1', x_2', ..., x_k')$ and $\mathbf{y}' = (y_1', y_2', ..., y_k')$ to represent their locations. Evidently, $x_i' = x_i + p \cdot W$ and $y_i' = y_i + q \cdot H$, where p and q are integers. The x and y components of the total intra-PE and inter-PE HPWL are denoted as L_{in}^x, L_{in}^y, L_{ξ}^x, and L_{ξ}^y, respectively. Then, the objective of the ILP is described by

$$\text{Minimize} \quad L_{in}^x(\mathbf{x}) + L_{in}^y(\mathbf{y}) + L_{\xi}^x(\mathbf{x}) + L_{\xi}^y(\mathbf{y}) \tag{1}$$

The HPWL functions are to be elaborated through their x components as the y components work in the same way. The x component of intra-PE wirelength is defined by

$$L_{in}^x(\mathbf{x}) = \sum_{N_i \in \mathcal{N}_{in}} \max_{b_j \in N_i, b_l \in N_i, j \neq l} |x_j - x_l| \tag{2}$$

and one example is net (b_1, b_3) in Figure 1. The x component of inter-PE wirelength is defined by

$$L_{\xi}^x(\mathbf{x}) = \sum_{N_i \in \mathcal{N}_{\xi}} \max_{b_j \in N_i, b_l' \in N_i} \begin{cases} x_l' - x_j, & \text{if } p > 0 \\ x_j - x_l', & \text{if } p < 0 \end{cases} \tag{3}$$

where $p > 0$ ($p < 0$) indicates block b_l' is on the right (left) of the reference PE like block b_1' (b_2') in Figure 1. The case of $p = 0$ is trivial. Although an inter-PE net may contain two blocks b_j and b_l, both of which are in the reference PE, their distance $|x_j - x_l|$ can never exceed the maximum inter-PE distance in the same net, and therefore does not need to be included in Equation (3). The max operations in Equations (2) and (3) are nonlinear and not supported by ILP solvers in general. This problem can be solved by introducing assist variables and additional constraints. For instance, $\max(z_1, z_2, ...)$ can be replaced by a new variable u that satisfies $u \geqslant z_1, u \geqslant z_2, ...$. As the new variables are to be minimized in the optimization, they become tight upper bounds or the maximum values. The nonlinear absolute value operation $|x_j - x_l|$ in the objective function can be replaced by $\max(x_j - x_l, x_l - x_j)$ and handled in the same manner.

The constraints in the ILP are the following.

$$|x_j - x_l| + |y_j - y_l| \geqslant 1, \ \ j, l = 1, 2, ..., k, \ \ j \neq l \tag{4}$$

$$x_j, y_j \in \mathbb{Z}, \ \ j = 1, 2, ..., k \tag{5}$$

$$x_{min} \leqslant x_j \leqslant x_{max}, \ \ y_{min} \leqslant y_j \leqslant y_{max}, \ \ j = 1, 2, ..., k \tag{6}$$

where x_{min}, x_{max}, y_{min} and y_{max} are given constants. Constraint (4) is to ensure that two different blocks do not occupy the same place. Constraints (5) and (6) make the solutions to be integers in legitimate ranges.

Although the absolute value $|x_j - x_l|$ in constraint (4) can be transformed to $\max(x_j - x_l, x_l - x_j)$, the max operation here cannot be simply replaced by an upper bound variable u like in handling $|x_j - x_l|$ for the objective function (2), which is to be minimized. Such upper bound variable would not be minimized in a constraint and hence cannot be tight as the maximum value.

We devise a technique to solve this problem and use operation $|z|$ as an example. This is a specific technique for an integer z in the context of ILP. A sufficiently large constant B is introduced as well as two additional integer variables e and $\hat{z}$. The value of $\hat{z}$ is forced to be equal to $|z|$ by adding

the following constraints in the ILP formulation.

$$z + Be \geqslant 0 \tag{7}$$
$$-z + B(1-e) \geqslant 0 \tag{8}$$
$$-z - B(1-e) \leqslant \hat{z} \leqslant z + Be \tag{9}$$
$$z - Be \leqslant \hat{z} \leqslant -z + B(1-e) \tag{10}$$
$$e \in \{0, 1\}$$
$$\hat{z} \in \mathbb{Z}$$

When $z > 0$, constraints (7) and (8) in the ILP solver would force $e = 0$. Then, Equations (9) and (10) become

$$-z - B \leqslant \hat{z} \leqslant z \tag{11}$$
$$z \leqslant \hat{z} \leqslant -z + B \tag{12}$$

which can be combined to $\hat{z} = z$.

When $z < 0$, constraints (7) and (8) in the ILP solver would force $e = 1$. Then, Equations (9) and (10) become

$$-z \leqslant \hat{z} \leqslant z + B \tag{13}$$
$$z - B \leqslant \hat{z} \leqslant -z \tag{14}$$

which can be combined to $\hat{z} = -z$. Therefore, $\hat{z} = |z|$.

Since the decision variables can be reduced to the positions of the blocks in the reference PE, the computational complexity of the ILP approach is decided by k, which is the number of blocks within a PE, and does not increase with the number of PEs ($m \cdot n$) in a given array.

C. Phase II: Customized Simulated Annealing

Phase II takes the PE array placement solution from phase I as part of its initial solution, and places the other blocks, which are not PEs, randomly in the initial solution. Then, simulated annealing is performed with acceleration by exploiting the regularity. The objective is to minimize total wirelength including intra-PE, inter-PE, PE-external and external wirelength. At the same time, the placement solution must be legal and does not have any overlap. In addition, the regular array structure is retained in PE placement. There are two kinds of moves in the simulated annealing: (1) moving a block to an empty legal space; (2) swapping the locations of two blocks of the same kind in terms of CLBs, RAMs and IO pads.

The simulated annealing here is considerably different from the conventional one due to the regularity and our customization. First, a move of a block in one PE must be repeated for the same corresponding blocks in the other PEs in an identical manner. Second, moves of PE blocks and non-PE blocks are searched with different levels of effort. Although the PE array placement in phase I is an approximation due to the neglection of PE-external nets and the assumption of homogeneous architecture substrate, its intra-PE and inter-PE wirelength has already been heavily optimized and the overall solution is far better than a random placement. As such, we explore much less number of moves for PE blocks than non-PE blocks. Moveover, the annealing temperature for PE block moves is set to be lower than that of non-PE blocks as only small changes are sufficient in general.

We notice that a very time consuming part of the simulated annealing is the cost function (total wirelength) estimation by VPR, which evaluates the impact of FPGA routing architectures. Based on this observation, we develop three techniques for further runtime reduction, and the tradeoff between runtime and solution quality: (1) speculative decision; (2) wirelength calculation omission; (3) hybrid wirelength model, which are to be elaborated as follows.

1) Speculative Decisions for PE Block Moves: One observation is that a block is preferred to be placed in boundary regions of a PE if it has many wire connections with circuit elements outside of the PE. Likewise, a block whose wire connections are mostly inside the PE is preferred to be placed in the middle of the PE. Therefore, we can sometimes quickly evaluate a simulated annealing move by examining the internal and external connections of PE blocks without the time consuming estimation of cost function, which is the wirelength computation considering FPGA routing architecture. Note that this does not conflict with the wirelength calculation omission, which is to be described in Section III-C2. The two techniques are applicable for different scenarios. The concept of internal/external connections has overlap with intra-PE/inter-PE nets defined in Section III-A, but is different. A net with h sinks is counted as h *connections* and internal connections are all within a single PE. External connections of a block b_i include connections with non-PE blocks in addition to inter-PE connections.

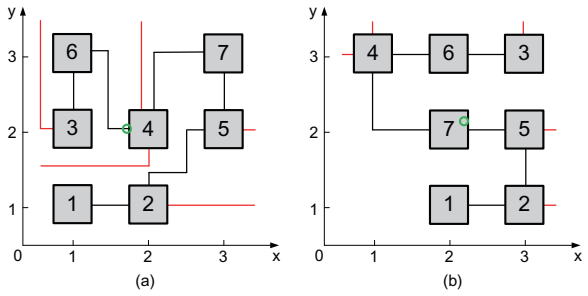


Fig. 2: Examples for calculating favorability score. Green circles indicate center of gravity. Black and red lines indicate internal and external connections, respectively.

The key ingredient for the internal/external connection-based speculation is a *favorability score* elaborated as follows. We reuse the concept of reference PE π_{ref} defined in Section III-B and assume that there are k blocks in each PE. Given a PE placement solution, the Center Of Gravity (COG) of its blocks is represented by $(\hat{x}, \hat{y})$, and the distance from COG to each block $b_i \in \pi_{ref}$ is denoted as d_i. Then, the favorability score is defined as

$$\phi = \sum_{i=1}^{k} d_i \cdot \frac{E_i}{I_i}$$

where I_i and E_i indicate the numbers of internal and external connections for block b_i, respectively. The score is high and favorable when a block with high E_i/I_i is far from the COG.

Examples of the favorability score calculation and its effect are illustrated in Figure 2. In Figure 2(a), the COG is at $(\frac{13}{7}, 2)$. Its favorability score is obtained by $\frac{E_2}{I_2} \times d_2 + \frac{E_3}{I_3} \times$

$d_3 + \frac{E_4}{I_4} \times d_4 + \frac{E_5}{I_5} \times d_5 = \frac{1}{2} \times 1.143 + \frac{1}{1} \times 0.857 + \frac{2}{2} \times 0.143 + \frac{1}{2} \times 1.143 = 2.143$. The favorability score for the placement in Figure 2(b) is estimated to be 5.071 through the same calculation procedure. One can see that placement in Figure 2(b) is more preferred than Figure 2(a).

Algorithm 1: Speculative decision for PE block move

Input: *PE placement solution after trial move μ;*
$I_i, i = 1, 2, ..., k$: #internal connections;
$E_i, i = 1, 2, ..., k$: #external connections;
$\tilde{\phi}$: favorability score before the move;
θ_L and θ_H: constant thresholds and $\theta_L < \theta_H$.

1 *PE center of gravity $\hat{x} = \sum_{i=1}^{k} x_i/k$, $\hat{y} = \sum_{i=1}^{k} y_i/k$;*
2 **for** *each block $b_i \in \pi_{ref}$* **do**
3 $\quad d_i = |x_i - \hat{x}| + |y_i - \hat{y}|$;
4 **end**
5 $\phi = \sum_{i=1}^{k} d_i \cdot \frac{E_i}{I_i}$; *// Favorability score*
6 **if** $\phi < \theta_L$ **then**
7 $\quad$ *Reject μ;*
8 **end**
9 **else if** $\phi > \theta_H$ *and* $\tilde{\phi} \leqslant \theta_H$ **then**
10 $\quad$ *Commit μ;*
11 **end**
12 **else**
13 $\quad$ *Evaluate wirelength to commit or reject μ;*
14 **end**

The speculative decision procedure is summarized in Algorithm 1. In step 1, the COG of the given placement solution after a trial move μ is calculated. The favorability score is obtained through steps 2-5. If the score is very low, which indicates a poor solution, move μ is directly rejected in step 7 without computing the cost function of simulated annealing. If the score is high while the solution prior to the move is not good enough, move μ is directly committed in step 10 without computing the cost function. Only in those unclear cases, the cost function is evaluated and a normal simulated annealing decision is made in step 13.

In our experience, the speculation is correct in most of time, but not always. Therefore, it can accelerate overall computing with a chance of small solution degradation.

2) Wirelength Calculation Omission for Repeating Patterns: This is a simple yet very effective technique that exploiting the repeating layout patterns in a PE array. It is illustrated with the example in Figure 3. There are 4 PEs $\pi^{1,1}$, $\pi^{1,2}$, $\pi^{2,1}$ and $\pi^{2,2}$. Due to the regularity, nets $(b_1^{1,1}, b_3^{1,1})$, $(b_1^{1,2}, b_3^{1,2})$, $(b_1^{2,1}, b_3^{2,1})$ and $(b_1^{2,2}, b_3^{2,2})$ are identical. Similarly, nets $(b_2^{1,1}, b_1^{1,2})$ $(b_2^{2,1}, b_1^{2,2})$ are the same. When block b_1 is moved upward from Figure 3(a) to (b), only the wirelength update for the reference PE ($\pi^{1,2}$) (red color in Figure 3(b)) needs to be actually calculated. The other wirelength changes (blue color in Figure 3(b)) can be taken from that of the reference PE and the calculation can be omitted. Although this technique is simple, it allows purely computing acceleration without any solution quality loss.

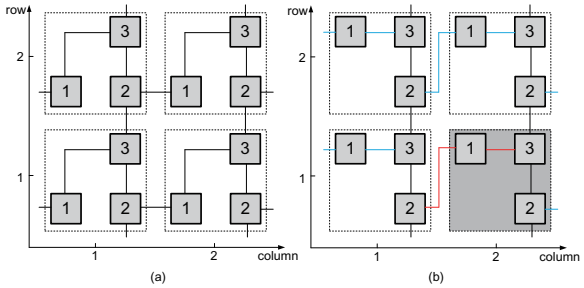


Fig. 3: When block b_1 is moved from (a) to (b), only wirelength associated with the reference PE (in red color) needs to be calculated, as the other wire patterns (in blue color) are the same.

3) Hybrid Wirelength Model: Besides HPWL, net wirelength can be estimated more accurately by considering FPGA routing architecture and VPR provides such pre-routing wirelength estimation function. However, calling this function costs more runtime than HPWL estimation. Therefore, there is runtime-accuracy tradeoff between HPWL and FPGA-aware wirelength estimation. Our idea is to use HPWL in early iterations of simulated annealing when an approximate global solution is searched and the FPGA-aware wirelength model is employed in later iterations when the solution search is mostly to refine details.

IV. Experimental Evaluations

A. Testcase Design

Conventional benchmark circuits for FPGA placement are not oriented toward designs with regularity. Hence, we designed a set of circuits with regularity as the testcases in the experiment. Here our choice on the regularity format is the systolic array, which is a widely used architecture topology in many important application domains, such as machine learning and digital signal processing. More specifically, considering that neural network circuit is currently the most representative and popular systolic array-based hardware application, we prepare testcases with focus on different types of systolic array-based neural network circuits, including Convolutional Neural Network (**CNN**) for computer vision, Multi-Layer Perceptron (**MLP**) for speech recognition, and Recurrent Neural Network (**RNN**) for natural language processing.

For each testcase, the systolic array consists of m-by-m PEs, where m is set as 4, 8, 16, 32, and 64 for each neural network type. Figure 4 is an example of 4×4 systolic array, where the top and left Random Access Memories (RAMs) are input RAMs, and the right RAMs are output RAMs. As a reference, the systolic array in Google TPU [26] design is 256×256. Each PE consists of one 16-bit multiplier, one 16-bit adder, two 8-bit registers and one 16-bit register (as seen in Figure 5). Notice that as the size of systolic array scales, the overall circuit complexity increases rapidly. For instance, a 64×64 systolic array testcase contains 4096 multipliers, 4096 adders and 12288 registers, which is a relatively large-size design

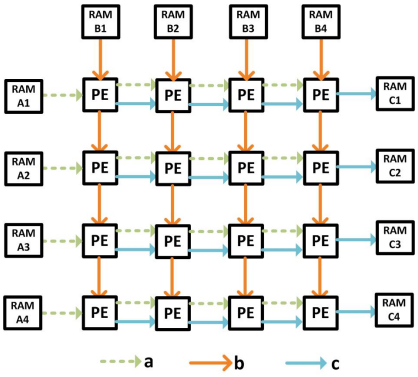


Fig. 4: An example design of 4×4 systolic array.

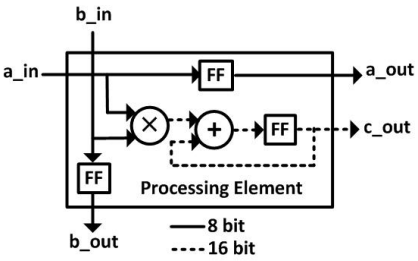


Fig. 5: One example design of a PE.

from the perspective of FPGA placement. Our testcase designs include 15 different neural network circuits with different PE array sizes.

B. Execution Flow

Given a Verilog file, logic synthesis is performed and verified in ODIN II [32]. VQM [33] and Quartus [29] are used to instantiate RAM blocks described in Verilog. The FPGA architecture is based on "*stratixiv_arch.timing.xml*", which is used in VPR *Titan* benchmarks. This architecture is close to a realistic one in the Altera Stratix FPGA Family, which has been applied for large circuit designs. In this architecture, columns of DSP blocks and block RAM are interleaved with CLB columns. The packing is performed and adjusted in VPR [5], after which our placement performed. The ILP solver used in our method is Gurobi 9.01 [34]. Finally, VPR takes our placement results for its routing and then evaluates post-routing wirelength as well as circuit timing.

The utilization for RNN circuits after packing is summarized in Table I, where LUT represents look-up table, and FF indicates flip-flop. Note that VPR supports customizing the size of an FPGA, so we use the absolute number of LUTs and FFs as the metric for FPGA utilization instead of the percentage. The utilizations of CNN and MLP circuits are similar. The circuit size for 64×64 systolic array is pretty large, but is still practical considering that high-end industrial FPGA product may contain more than three millions of LUTs [35].

TABLE I: FPGA utilization of RNN circuits.

Array Size	#LUT	#FF
4×4	7874	464
8×8	35087	1920
16×16	160160	8192
32×32	576450	33280
64×64	2031106	135509

C. Runtime Improvement

Our placement method is implemented in C language. The experiments are performed on a Linux work station with an Intel 3.70GHz i5-9600K core and 16GB RAM. Our placement is compared with VPR 8.0.0 [5], which is perhaps the most influential academic tool. The results are showed in Table II. The speedup from our method increases with PE array size and more than 25× acceleration is achieved for neural networks with 64×64 systolic array. This speedup implies that the placement runtime for the large cases is reduced from about 13 hours to a little more than a half hour. It is significantly more than the 7× speedup obtained by an analytical approach [12]. Since runtime reduction is mostly needed for large designs, the increasingly large speedup is very appealing.

TABLE II: Results on placement runtime. (RNN: recurrent neural network; CNN: convolutional neural network; MLP: multilayer perceptron.)

Cases	Array Size	RAM Size	Runtime (s)		
			VPR	Ours	Speedup
RNN	4×4	256KB	318	127	2.5×
RNN	8×8	256KB	995	194	5.1×
RNN	16×16	256KB	2938	288	10.2×
RNN	32×32	256KB	9867	514	19.2×
RNN	64×64	256KB	46828	1653	28.3×
CNN	4×4	256KB	347	162	2.1×
CNN	8×8	256KB	1087	226	4.8×
CNN	16×16	256KB	3233	327	9.9×
CNN	32×32	256KB	9894	481	20.6×
CNN	64×64	256KB	50327	1792	28.1×
MLP	4×4	512KB	582	343	1.7×
MLP	8×8	512KB	1662	404	4.1×
MLP	16×16	512KB	4214	487	8.3×
MLP	32×32	512KB	10253	585	17.5×
MLP	64×64	512KB	51197	2014	25.4×

D. Solution Quality and Correlations

The routing for all the placement solutions can be easily completed by VPR. The post-routing wirelength and circuit frequency results are shown in Figure 6. These results are normalized using VPR results as the baseline. On average, our method results in 6.9% wirelength increase and 6.8% circuit frequency decrease. A main reason for the degradation is from the dead space caused by maintaining regularity. Without the regularity constraint, VPR is able to find more compact solutions. Therefore, this degradation is a price paid for the regularity-based acceleration and difficult to be avoided.

We also investigate the correlation between our placement and VPR. The wirelength and frequency correlation results are shown in Figure 7 and Figure 8, respectively. In both figures,

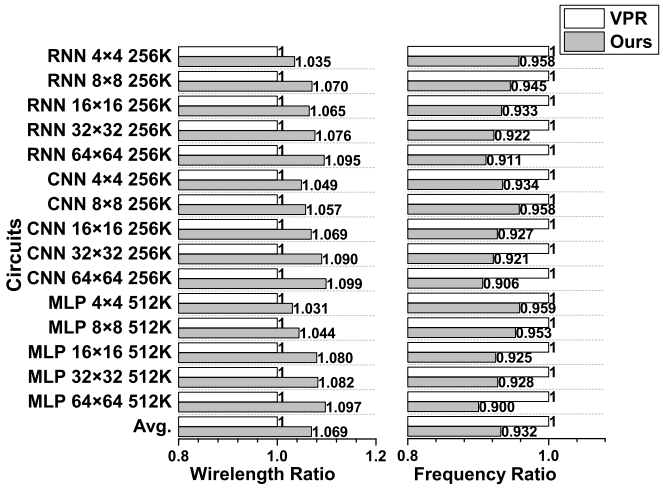


Fig. 6: Normalized wirelength (left) and frequency (right) from solutions of our placer and VPR.

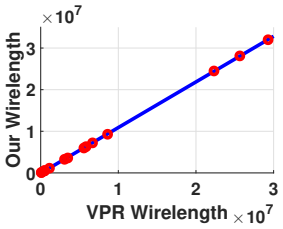


Fig. 7: Wirelength correlation between our solutions and VPR's, correlation coefficient 1.0000.

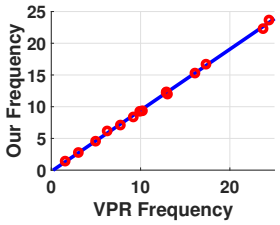
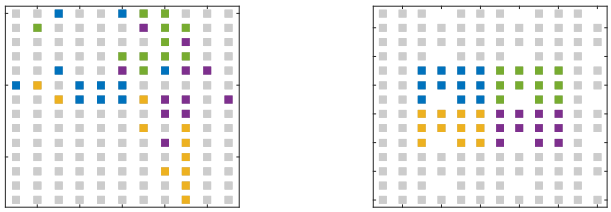


Fig. 8: Frequency correlation between our solutions and VPR's, correlation coefficient 0.9994.

each red circle indicates a data point, of which the x and y coordinates are the results from VPR and our placement for the same testcase, respectively. The blue lines are the linear regression results of the data points. For both wirelength and frequency, the results from our placement and VPR are nearly perfectly correlated, with the correlation coefficients being 1.0000 for wirelength and 0.9994 for frequency. The near perfect correlation allows the proposed method to serve for fast early prototyping.

E. Placement Solution Demonstration

A couple of placement solutions are demonstrated in Figure 9. Each figure shows the middle region within a 64×64 systolic array layout. Four structurally adjacent PEs are highlighted with colors and all blocks of the same PE have the same color. Blocks of the other PEs are represented by gray squares. The regularity from our placement is evident in Figure 9(b) while VPR cannot retain the regularity as shown in Figure 9(a). On the other hand, the regularity results in vacant places, which can be seen in Figure 9(b). Therefore, our placement is not as compact as VPR and this explains the wirelength increase from our placement.



(a) VPR (b) Ours

Fig. 9: Placement of 4 PEs in a 64×64 array.

F. Analysis of Individual Techniques

Besides maintaining regularity and thus reducing decision variables in simulated annealing, four other techniques are developed and included in our proposed method.

- **ILP**: PE array placement by ILP (Section III-B) as a part of the initial solution for simulated annealing.
- **Speculation**: Speculative decisions for PE block moves described in Section III-C1.
- **Omission**: The wirelength calculation omission proposed in Section III-C2.
- **Hybrid**: Using hybrid HPWL and VPR wirelength model as described in Section III-C3.

For ILP, one particularly appealing feature is that its complexity is decided by PE size and independent of the number of PEs. There are hundreds of variables in the ILP formulation for the testcases and most of them are associated with the transformations in handling the absolute values. The number of constraints is over 1000. The measured runtime versus PE array size is plotted in Figure 10, which confirms its independence of the number of PEs.

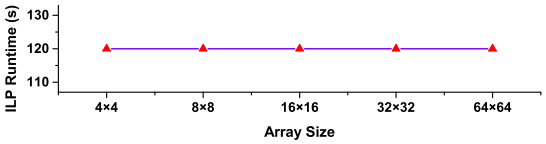


Fig. 10: ILP runtime versus PE array size.

The effect of each individual techniques are further analyzed through experiments with results shown in Figures 11, 12 and 13. The effects are demonstrated through comparisons with the following.

- **Baseline1**: None of the 4 techniques is used and wirelength is completely evaluated by HPWL model in simulated annealing.
- **Baseline2**: None of the 4 techniques is used and wirelength is completely evaluated by VPR model in simulated annealing.
- **All**: All of the 4 techniques are applied.

In Figures 11, 12 and 13, the results for one technique means only this technique is applied on top of Baseline1.

The runtime speedup comparisons are shown in Figure 11. One can see that ILP and Omission contribute to most of the

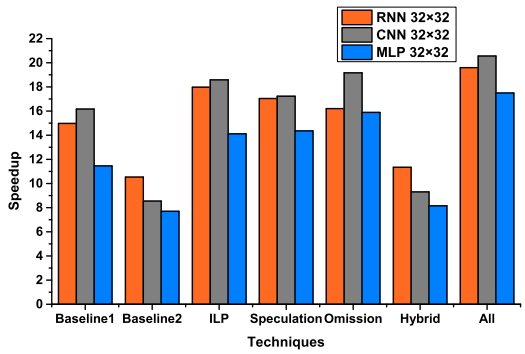


Fig. 11: The effect of each technique on runtime speedup.

speedup among the 4 techniques and the speedup improvement from Speculation is also close. ILP and Speculation reduce the time on wirelength estimation at the expense of solution degradation. Hybrid is intended to tradeoff between speedup and solution quality and therefore its speedup is lower than Baseline1 while it is still higher than Baseline2.

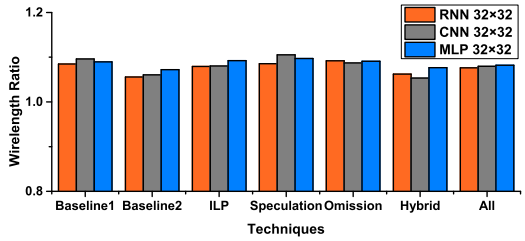


Fig. 12: The effect of each technique on wirelength.

The effects on wirelength are compared in Figure 12 where the vertical axis corresponds to normalized wirelength with respect to VPR solutions. ILP and Hybrid can reduce wirelength overhead a little compared to Baseline1. The Omission wirelength is supposed to be the same as that of Baseline1 in theory. The slight difference is due to the randomness in simulated annealing. Overall, the impact to wirelength from these techniques vary little. This observation implies that the wirelength increase from our method is mostly due to the regularity constraints instead of algorithm deficiency.

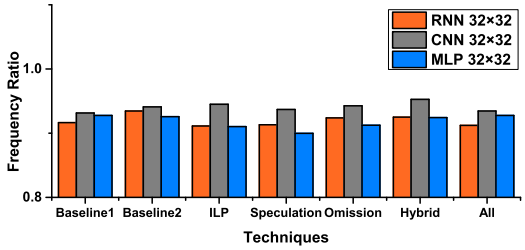


Fig. 13: The effect of each technique on frequency.

The frequency results are depicted in Figure 13, where the vertical axis indicates normalized frequency with respect to VPR results. Among these techniques, Hybrid has the minimum frequency degradation while the frequencies from the other techniques are close to each other. This trend also indicates that the frequency degradation is more from regularity constraints than the algorithms.

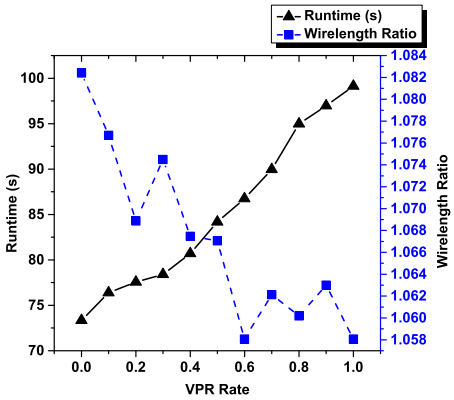


Fig. 14: Runtime and wirelength tradeoff by different VPR wirelength model use rate on an 8×8 systolic array.

In the Hybrid technique, HPWL model is used in the first $1-\rho$ portion of simulated annealing iterations, where $\rho \in [0, 1]$ is a parameter, and VPR wirelength model is employed in the rest ρ portion. The rate ρ of using VPR model affects the tradeoff between runtime and solution quality. Experiments are performed on an 8×8 systolic array for studying this tradeoff and the results are plotted in Figure 14. As the use of VPR model increases, the simulated annealing runtime increases monotonically. At the same time, wirelength decreases in general with some kinks due to the heuristic nature of the algorithm as well as model inaccuracy.

V. Conclusions and Future Research

There is a strong need for fast FPGA placement and increasing popularity for designs with regularity. This work is an effort to leverage the latter for helping the former. At the same time, the pros and cons of this approach are evaluated. A hybrid ILP and simulated annealing-based FPGA placement method is developed for design regularity. It achieves more than $25\times$ speedup versus a classical academic tool on relatively large neural network designs on FPGA. Meanwhile, it pays a price of near 10% wirelength and circuit frequency degradation. On the other hand, it has almost perfect correlation with the classical method. The proposed method will be useful for designs with tight turn-around time and early design prototyping.

In future research, analytical FPGA placement considering regularity will be investigated. Furthermore, other design objectives, such as routability and clock domains, will also be addressed.

REFERENCES

[1] A. Putnam, A. M. Caulfield, E. S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. P. Gopal, J. Gray, M. Haselman, S. Hauck, S. Heil, A. Hormati, J.-Y. Kim, S. Lanka, J. Larus, E. Peterson, S. Pope, A. Smith, J. Thong, P. Y. Xiao, and D. Burger, "A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services," *IEEE Micro*, vol. 35, no. 3, pp. 10–22, 2015.

[2] M. Motamedi, P. Gysel, V. Akella, and S. Ghiasi, "Design Space Exploration of FPGA-Based Deep Convolutional Neural Networks," *ACM/IEEE Asia and South Pacific Design Automation Conference*, pp. 575–580, 2016.

[3] H. Zheng, S. T. Gurumani, K. Rupnow, and D. Chen, "Fast and Effective Placement and Routing Directed High-Level Synthesis for FPGAs," *ACM International Symposium on Field-Programmable Gate Arrays*, p. 1–10, 2014.

[4] V. Betz and J. Rose, "VPR: A New Packing, Placement and Routing Tool for FPGA Research," *International Workshop on Field-Programmable Logic and Applications*, pp. 213–222, 1997.

[5] J. Luu, J. Goeders, M. Wainberg, A. Somerville, T. Yu, K. Nasartschuk, M. Nasr, S. Wang, T. Liu, N. Ahmed, K. B. Kent, J. Anderson, J. Rose, and V. Betz, "VTR 7.0: Next Generation Architecture and CAD System for FPGAs," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 7, no. 2, pp. 1–30, 2014.

[6] S. Dai, Y. Zhou, H. Zhang, E. Ustun, E. F. Young, and Z. Zhang, "Fast and Accurate Estimation of Quality of Results in High-Level Synthesis with Machine Learning," *IEEE International Symposium on Field-Programmable Custom Computing Machines*, pp. 129–132, 2018.

[7] Y. Sankar and J. Rose, "Trading quality for compile time: ultra-fast placement for FPGAs," in *ACM International Symposium on FPGA*, pp. 157–166, 1999.

[8] M. G. Wrighton and A. M. DeHon, "Hardware-Assisted Simulated Annealing with Application for Fast FPGA Placement," *ACM International Symposium on Field Programmable Gate Arrays*, pp. 33–42, 2003.

[9] Y. Xu and M. A. Khalid, "QPF: Efficient Quadratic Placement for FPGAs," *International Conference on Field Programmable Logic and Applications*, pp. 555–558, 2005.

[10] A. Choong, R. Beidas, and J. Zhu, "Parallelizing Simulated Annealing-Based Placement Using GPGPU," *International Conference on Field Programmable Logic and Applications*, pp. 31–34, 2010.

[11] M. Gort and J. H. Anderson, "Analytical Placement for Heterogeneous FPGAs," *International Conference on Field Programmable Logic and Applications*, pp. 143–150, 2012.

[12] T.-H. Lin, P. Banerjee, and Y.-W. Chang, "An Efficient and Effective Analytical Placer for FPGAs," *ACM/IEEE Design Automation Conference*, pp. 1–6, 2013.

[13] M. An, J. G. Steffan, and V. Betz, "Speeding Up FPGA Placement: Parallel Algorithms and Methods," *IEEE International Symposium on Field-Programmable Custom Computing Machines*, pp. 178–185, 2014.

[14] R. Pattison, Z. Abuowaimer, S. Areibi, G. Grewal, and A. Vannelli, "GPlace: A Congestion-Aware Placement Tool for UltraScale FPGAs," *IEEE/ACM International Conference on Computer-Aided Design*, pp. 1–7, 2016.

[15] G. Chen, C. Pui, W. Chow, K. Lam, J. Kuang, E. F. Y. Young, and B. Yu, "RippleFPGA: Routability-Driven Simultaneous Packing and Placement for Modern FPGAs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 10, pp. 2022–2035, 2018.

[16] W. Li and D. Z. Pan, "A New Paradigm for FPGA Placement Without Explicit Packing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 11, pp. 2113–2126, 2019.

[17] J. Yuan, J. Chen, L. Wang, X. Zhou, Y. Xia, and J. Hu, "ARBSA: Adaptive Range-Based Simulated Annealing for FPGA Placement," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 12, pp. 2330–2342, 2019.

[18] A. Koch, "Structured Design Implementation - A Strategy for Implementing Regular Datapaths on FPGAs," *ACM International Symposium on Field-Programmable Gate Arrays*, pp. 151–157, 1996.

[19] C. Lavin and A. Kaviani, "RapidWright: enabling custom crafted implementations for FPGAs," in *IEEE International Symposium on Field-Programmable Custom Computing Machines*, pp. 133–140, 2018.

[20] N. Zhang, X. Chen, and N. Kapre, "RapidLayout: fast hard block placement of FPGA-optimized systolic arrays using evolutionary algorithms," in *IEEE International Conference on Field Programmable Logic and Applications*, 2020.

[21] P. S. Kumar and Z. David, *Neural Networks and Systolic Array Design*, vol. 49. World Scientific, 2002.

[22] C. R. Castro-Pareja, J. M. Jagadeesh, S. Venugopal, and R. Shekhar, "FPGA-Based 3D Median Filtering Using Word-Parallel Systolic Arrays," *IEEE International Symposium on Circuits and Systems*, vol. 3, pp. III–157, 2004.

[23] Y. Yang, W. Zhao, and Y. Inoue, "High-Performance Systolic Arrays for Band Matrix Multiplication," *IEEE International Symposium on Circuits and Systems*, pp. 1130–1133, 2005.

[24] P. K. Meher, "Efficient Systolic Implementation of DFT Using a Low-Complexity Convolution-Like Formulation," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 53, no. 8, pp. 702–706, 2006.

[25] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, 2016.

[26] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P.-l. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon, "In-Datacenter Performance Analysis of a Tensor Processing Unit," *International Symposium on Computer Architecture*, pp. 1–12, 2017.

[27] X. Wei, C. H. Yu, P. Zhang, Y. Chen, Y. Wang, H. Hu, Y. Liang, and J. Cong, "Automated Systolic Array Architecture Synthesis for High Throughput CNN Inference on FPGAs," *ACM/IEEE Design Automation Conference*, pp. 1–6, 2017.

[28] J. Zhang, W. Zhang, G. Luo, X. Wei, Y. Liang, and J. Cong, "Frequency Improvement of Systolic Array-Based CNNs on FPGAs," *IEEE International Symposium on Circuits and Systems*, pp. 1–4, 2019.

[29] Intel Quartus. https://www.intel.com/content/www/us/en/programmable/documentation/zpr1513988353912.html.

[30] J. Cong, B. Liu, S. Neuendorffer, J. Noguera, K. Vissers, and Z. Zhang, "High-Level Synthesis for FPGAs: From Prototyping to Deployment," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 4, pp. 473–491, 2011.

[31] L. Liu, J. Weng, and N. Kapre, "RapidRoute: Fast Assembly of Communication Structures for FPGA Overlays," *IEEE International Symposium on Field-Programmable Custom Computing Machines*, pp. 61–64, 2019.

[32] P. Jamieson, K. B. Kent, F. Gharibian, and L. Shannon, "Odin II - An Open-Source Verilog HDL Synthesis Tool for CAD Research," *IEEE Annual International Symposium on Field-Programmable Custom Computing Machines*, pp. 149–156, 2010.

[33] K. E. Murray, S. Whitty, S. Liu, J. Luu, and V. Betz, "From Quartus to VPR: Converting HDL to BLIF with the Titan flow," *International Conference on Field programmable Logic and Applications*, pp. 1–1, 2013.

[34] Gurobi Optimization. http://www.gurobi.com.

[35] Xilinx Versal Architecture and Product Data Sheet. https://www.xilinx.com/support/documentation/data_sheets/ds950-versal-overview.pdf.