Global Placement Exploiting Soft 2D Regularity

Donghao Fang
Texas A&M University
donghao@tamu.edu

Boyang Zhang
Rutgers University
bz207@scarletmail.rutgers.edu

Hailiang Hu
Texas A&M University
hailiang@tamu.edu

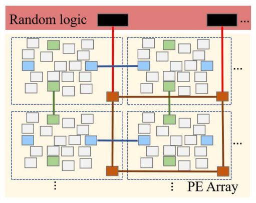
Wuxi Li
Xilinx
wuxili@xilinx.com

Bo Yuan
Rutgers University
bo.yuan@soe.rutgers.edu

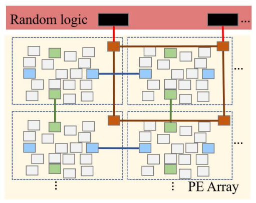
Jiang Hu
Texas A&M University
jianghu@tamu.edu

ABSTRACT

Cell placement is such a critical step for chip physical design that it needs many kinds of efforts for improvement. Recently, designs with 2D processing element arrays have become popular primarily due to their deep neural network computing applications. The 2D array regularity is similar to but different from the regularity of conventional datapath designs. To exploit the 2D array regularity, this work develops a new global placement technique built upon RePlAce, the latest state-of-the-art placement framework. Experimental results from various designs show that the proposed technique can reduce half-perimeter wirelength and Steiner tree wirelength by about 6% and 12%, respectively.



(a) PE cells are placed in macro generation, wirelength: 264



(b) PE cells are placed along with random logic, wirelength: 244

Figure 1: Macro-based design versus flat design.

1 INTRODUCTION

Cell placement is undoubtedly a step of paramount importance in chip physical design. As such, people have spent relentless research efforts to improve it for general cases as well as commonly seen exceptional cases. One recently popular case is circuit design with two-dimensional processing element (PE) arrays, such as systolic arrays [Quinton 1987]. The popularity is mainly due to their applications in convolutional neural network computing, e.g., Google TPU [Jouppi et al. 2017] and Eyeriss [Chen et al. 2019]. Systolic arrays also have many other applications such as signal processing [Chan and Chen 1988] and communication circuit designs [Asai and Matsumoto 2000].

In general, there can be two approaches to the placement of a design with a 2D PE array: (1) cell placement of a PE is generated in advance and used as macros at chip-level integration, where macro placement can be obtained through either manual designs or software tools; (2) flat design where cells of PEs and random logic are simultaneously placed. While the former approach is conceivably faster, the latter one potentially delivers better solution quality. During cell placement of a PE in macro generation, information about random logic placement is generally unavailable. Later, when placing macros along with random logic, the relative locations of cells in a PE are fixed. Therefore, in the macro-based approach, it is challenging for intra-PE cell placement to minimize the wirelength of nets connecting PE cells and random logic cells. By contrast, such nets can be easily handled during flat design for wirelength reduction. Since a PE typically has more than a hundred cells, the impact of intra-PE cell placement can be significant. In the example of Figure 1(a), PE cells are placed during macro generation without knowledge of random logic locations. In Figure 1(b), PE cells are placed along with random logic in a flat design. If the example is for a 4×4 PE array, the flat design wirelength is about 8% shorter than the macro-based design.

Although general-purpose placement tools are capable of simultaneous placement of PE and random logic cells, they are oblivious to the regularity of PE arrays. From more than 30 years ago, it was found that regularity can facilitate better placement solutions [Cai et al. 1990]. There have numerous studies on regularity-driven placement [Cai et al. 1990; Chou et al. 2012; Nijssen and Jess 1996; Ward et al. 2013; Yang et al. 2003; Ye and De Micheli 2000]. However, almost all the previous works are focused on conventional datapath designs, whose regularity is repeated bit-slice patterns. The regularity of 2D PE arrays is significantly different from that of conventional datapath, and the difference is discussed with more details in Section 4. Although the benefit of utilizing 2D PE array regularity has been demonstrated for FPGA layout [Kong et al. 2020; Zhang et al. 2019], there is no previous work on ASIC cell placement with dedicated treatment to the regularity of 2D PE arrays, to the best of our knowledge.

In this work, we investigate how to exploit the regularity of 2D PE arrays for reducing wirelength of placement solutions. We assume that regularity has already been extracted using techniques similar to previous work [Nijssen and Jess 1996]. Our work is built upon RePlAce [Cheng et al. 2019], which is the latest state-of-the-art for global placement. In the proposed technique, PE and random logic cells are placed simultaneously. The contributions of this work include the following.

- This work proposes the first ASIC global placement technique exploiting the regularity of 2D PE arrays, to the best of our knowledge.
- Experiments are performed on matrix multiplication and neural network circuits, including cases with over 1 million cells. Compared to RePlAce, the proposed technique reduces half-perimeter wirelength and Steiner tree wirelength by about 6% and 12%, respectively, with a limited runtime increase. Its half-perimeter wirelength is also about 11% smaller than NTUplace3 [Chen et al. 2008] and POLAR [Lin et al. 2013].

The rest of this paper is organized as follows. The background on systolic array (2D PE array) is introduced in Section 2. Previous related works are summarized in Section 3. The difference between the regularity of 2D PE arrays and conventional datapath is discussed in Section 4. Section 5 provides a brief review of RePlAce. Our regularity-aware placement technique is described in Section 6. Experimental results are shown in Section 7, which is followed by the conclusion and future research in Section 8.

2 SYSTOLIC ARRAY AND 2D PE ARRAY

A systolic array is an array of PEs (Processing Elements) for massively parallel computing. A PE is often composed of a MAC (Multiply-Accumulate) unit and registers. It has two signature properties: regularity and local data interconnect. A 2D systolic array for matrix multiplication $C = A \times B$ is demonstrated in Figure 2, where elements of matrices A and B are fed from the left and top, respectively, and the output matrix elements are accumulated locally at each PE. Data interconnects, which are multi-bit wide, are restricted between neighboring PEs. Such local interconnections avoid long signal propagation delays and thereby facilitate better performance. Local interconnects are also more routable than global ones. Since data move from each PE to its neighboring PEs like pulsation, such an array is called systolic array.

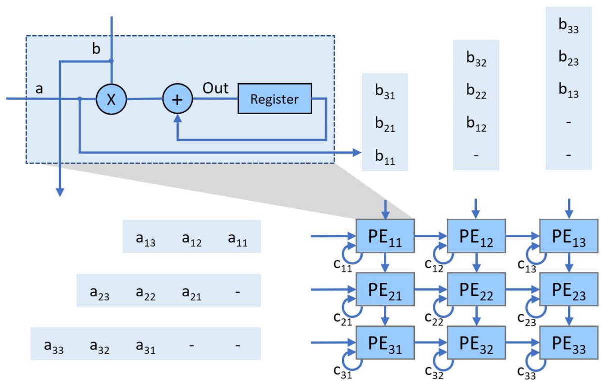

b
a
X
+
Out
Register
b_{31} b_{21} b_{11}
b_{32} b_{22} b_{12} -
b_{33} b_{23} b_{13} - -
a_{13} a_{12} a_{11}
a_{23} a_{22} a_{21} -
a_{33} a_{32} a_{31} - -
PE_{11} PE_{12} PE_{13}
PE_{21} PE_{22} PE_{23}
PE_{31} PE_{32} PE_{33}
c_{11} c_{12} c_{13}
c_{21} c_{22} c_{23}
c_{31} c_{32} c_{33}

Figure 2: A 2D systolic array for matrix multiplication.

The concept of systolic array was developed in the 1970s and became a hot research topic in the 1980s [Quinton 1987]. It has wide applications, including digital signal processing [Chan and Chen 1988], communication circuits [Asai and Matsumoto 2000], linear algebra computing [Vucha and Rajawat 2011], finite field division [Ibrahim et al. 2018] and hidden Markov model [Peltenburg et al. 2016], to name a few. Recently, it regained popularity mainly due to its applications in Convolutional Neural Network (CNN) computing [Jouppi et al. 2017; Wei et al. 2017; Zhang et al. 2019]. In theory, a systolic array can be 1D or higher than 2D. This work is focused on 2D as it is the most typical case in CNN computing. In some CNN hardware designs [Chen et al. 2019], specific data are broadcasted to PEs instead of going through local interconnects. Although such designs are not strictly systolic arrays, they still use 2D PE arrays. Thus, our technique is designed for 2D PE arrays, which are more general than systolic arrays.

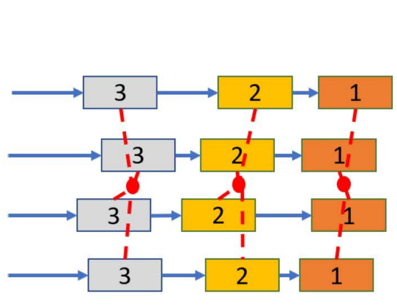
3 PREVIOUS RELATED WORK

Previous works on regularity-aware placement were mostly geared toward datapath designs, whose regularity refers to repeated bit-slice patterns. In [Cai et al. 1990], the repeated patterns allow datapath placement to be formulated and solved as a linear placement problem. An automatic regularity extraction technique is introduced in [Nijssen and Jess 1996]. An abstract physical model [Ye and De Micheli 2000] is proposed to capture the regularity to be utilized by the linear placement of datapath. In [Yang et al. 2003], soft alignment constraints are applied to a quadratic placement technique for datapath. Later, the interactions between datapath and random logic placement is considered in [Chou et al. 2012]. It first clusters datapath into large macros while cell locations within the macros are not decided. The macros are placed along with random logic through mixed-size placement. Since datapath cells have not been placed within the macros yet, the wirelength estimation for nets between datapath cells and random logic can be wildly inaccurate. Next, datapath is placed using nonlinear programming with hard alignment constraints while random logic cells are fixed. Last, random logic placement is refined while datapath placement is fixed. A simultaneous datapath and random logic placement method based on SimPL [Kim et al. 2011] is described in [Ward et al. 2013], where soft alignment constraints are applied to facilitate placement regularity. An integrated datapath extraction and placement method based on [Ward et al. 2013] is proposed in [Ward et al. 2012].

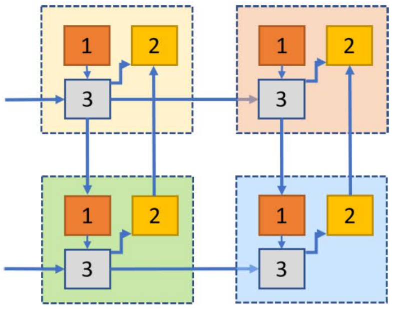
To the best of our knowledge, there is no previous work considering the regularity of 2D PE arrays for general cell placement apart from FPGAs. One remotely related work is [He et al. 2021], which is a floorplanning method for mapping CNN circuits onto wafer-scale processors. In an FPGA-based systolic array design [Zhang et al. 2019], each PE is constrained into a region identified by floorplanning so that the CNN circuit can run with improved frequency. The regularity of 2D PE arrays is utilized for FPGA placement in [Kong et al. 2020]. It achieves $2\times - 28\times$ speedup with $3\% - 9\%$ wirelength increase. However, its main techniques are simulated annealing and ILP (Integer Linear Programming), which are difficult to handle large placement problems in modern ASIC designs.

4 2D PE ARRAY VS. CONVENTIONAL DATAPATH

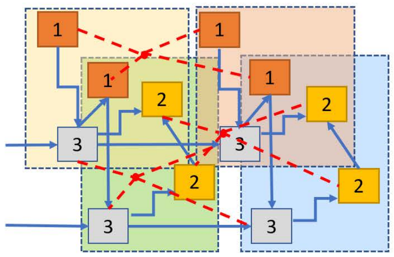
Despite the similarity, there is a significant difference between the regularity of 2D PE arrays and that of conventional datapath designs. In datapath designs, blocks in a bit slice form a contiguous row, while blocks of a bit stack, a.k.a. alignment group, can be aligned into a contiguous column. By contrast, the regularity of a PE array is coarse-grained and not continuous. More specifically, the repeated patterns among cells are interleaved across multiple PEs instead of being contiguous. For example, in Figure 3 (b), dark orange cells labeled with one repeat the same pattern, but they are interleaved with grey cells labeled with 3. Due to this difference, regularity techniques for datapath may not be applicable for a PE array. For example, in [Ward et al. 2013], a pseudo net is applied to each alignment group so that its blocks can be placed together. This is illustrated in Figure 3 (a), where blocks of the same alignment group have the same color. If this technique is directly applied for a PE array, a pseudo net is enforced to cells of the same regularity group (cells of the same color in Figure 3 (b) and (c)). Such pseudo nets would cause PE overlap, which is not preferred, as shown in Figure 3 (c).



(a) Pseudo nets for bit stacks of datapath



(b) An example of 2D PE array



(c) Applying pseudo nets on the 2D PE array

Figure 3: Different regularities between conventional datapath and 2D PE array. Red dashed lines indicate pseudo nets. Cells of the same color are in the same alignment/regularity group.

5 BACKGROUND ON REPLACE

RePlAce[Cheng et al. 2019] is the latest version of the electrostatics-based placement approach, called ePlace[Lu et al. 2015]. It includes enhancements to ePlace[Lu et al. 2015] and the capability of addressing routability. We mainly use its wirelength and cell density optimization techniques and defer the routability part into future work. RePlAce[Cheng et al. 2019] significantly outperforms almost all of its prior works, and the validation has been performed on almost all public domain benchmark suites, including those from ISPD, ICCAD, and DAC contests.

Here we briefly review some key concepts in RePlAce/ePlace to help understand our approach while details can be found in [Cheng et al. 2019; Lu et al. 2015]. Let $\boldsymbol{v} = (\boldsymbol{x}, \boldsymbol{y})^T$ denote the vectors of x-y coordinates of movable cells. The problem formulation is

$$\min_{\boldsymbol{v}} W(\boldsymbol{v}) + \lambda \cdot D(\boldsymbol{v}) \tag{1}$$

where $W(\boldsymbol{v})$ is the wirelength function, $D(\boldsymbol{v})$ is the cell density function and λ is a weighting factor. The wirelength function $W(\boldsymbol{v})$ attempts to estimate HPWL (Half Perimeter Wire Length) and is smoothed by a weighted average (WA) net model. The x-component of HPWL for net e in the WA model is given by

$$\widetilde{\mathrm{W}}_{e_x}(\boldsymbol{v}) = \frac{\sum_{i \in e} x_i \exp(x_i/\gamma)}{\sum_{i \in e} \exp(x_i/\gamma)} - \frac{\sum_{i \in e} x_i \exp(-x_i/\gamma)}{\sum_{i \in e} \exp(-x_i/\gamma)} \tag{2}$$

where γ is a parameter controlling the modeling accuracy.

Minimizing $D(\boldsymbol{v})$ is to spread out cells and reduce cell overlaps. A layout area is tessellated to a uniform grid of rectangle bins, and $D(\boldsymbol{v})$ quantifies the degree of cell overflows in this bin grid. A key innovation of RePlAce/ePlace is to model each cell with electric charges proportional to its area. Then, the repelling forces among the charges naturally spread out cells. Assuming that the bins form a $B \times B$ array, RePlAce/ePlace models $D(\boldsymbol{v})$ with a potential energy function

$$\Phi(\hat{x}, \hat{y}) = \sum_{0 \le \hat{x} < B} \sum_{0 \le \hat{y} < B} \rho(\hat{x}, \hat{y}) \psi(\hat{x}, \hat{y}) \tag{3}$$

where $\hat{x}/\hat{y}$ means column/row indices of bins, ρ is the charge density of a bin and $\psi(\hat{x}, \hat{y})$ represents local electric potential. The gradient $\nabla\Phi_v$ models the e-force (electric force) that repels cells away from each other. It is shown in [Lu et al. 2015] that $\psi(\hat{x}, \hat{y})$ can be computed from $\rho(\hat{x}, \hat{y})$ through discrete cosine transformation.

Then, the nonlinear programming problem becomes

$$\min_{\boldsymbol{v}} \tilde{W}(\boldsymbol{v}) + \lambda \cdot \Phi(\boldsymbol{v}) \tag{4}$$

where $\tilde{W}(\boldsymbol{v})$ is the WA model for the total HPWL. This problem is solved by Nesterov's method with some enhancements, including area-driven preconditioning and Lipschitz constant-based step size estimation. Compared to ePlace, RePlAce also applies a local density function into the objective for further improvement.

6 PLACEMENT WITH 2D PE ARRAY REGULARITY

6.1 Preliminaries

6.1.1 *Problem Formulation.* The input to placement is a netlist $G(V, E)$, where V denotes the set of movable cells and fixed IO cells, and E represents the set of nets. The netlist contains a 2D PE array, which has been identified by an automatic method such as [Nijssen and Jess 1996]. It also includes random logic such as buffers, controllers, etc. A legitimate assumption is that IO cells of the same type of data are placed contiguously on the same side of the layout area. The problem formulation is to minimize total wirelength subject to the constraint that there is no overlap among cells. Like in many previous works, HPWL is employed here for wirelength estimation. Please note that regularity is not a constraint in the problem formulation. Instead, it is a help for achieving small wirelength.

6.1.2 *Overview of the Proposed Approach.* Exploiting the 2D array regularity in RePlAce is not straightforward. Actually, directly enforcing regularity constraints throughout RePlAce makes solution convergence very difficult. We propose an approach whose overview is provided in Figure 4. In the Figure, white boxes indicate techniques from previous works, while colored boxes mean new changes by our approach. Step 0 is to determine the orientation of the given PE array. This part is essential for considering regularity in later steps and elaborated in Section 6.2. Step 1 is the initial quadratic placement like RePlAce [Cheng et al. 2019] and ePlace [Lu et al. 2015]. The difference here is that hard regularity constraints are enforced for PE array cells, and the details are provided in Section 6.3. Step 2 consists of kernel RePlAce iterations with array regularization and is described in Section 6.4. In Step 3, the original RePlAce iterations are continued. We adopt the detailed placer of NTUplace3 [Chen et al. 2008] for Step 4 in the proposed approach.

6.1.3 *Soft Regularity.* Since regularity is not considered in Step 3, the regularity in final solutions is soft instead of strict. Applying

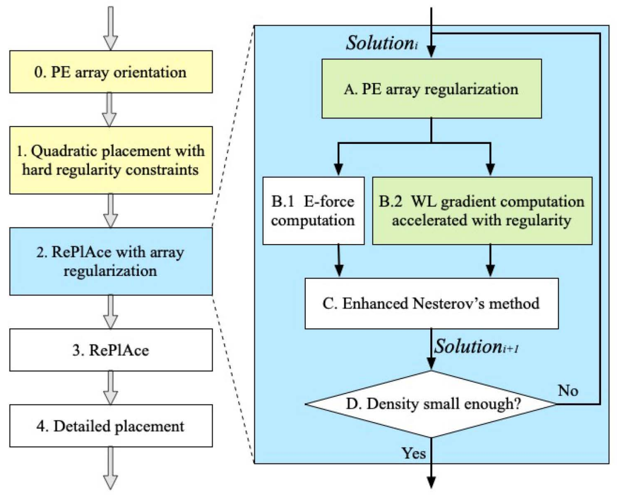

0. PE array orientation
1. Quadratic placement with hard regularity constraints
2. RePlAce with array regularization
3. RePlAce
4. Detailed placement
Solution$_i$
A. PE array regularization
B.1 E-force computation
B.2 WL gradient computation accelerated with regularity
C. Enhanced Nesterov's method
Solution$_{i+1}$
D. Density small enough?
No
Yes

Figure 4: Overview of the proposed approach.

regularity in Steps 1 and 2 can help better solutions. However, continuing to exercise regularity in later iterations causes struggles with the original RePlAce objective and makes convergence very difficult. A similar observation [Ward et al. 2013] was made for datapath regularity as well. In FPGA placement, hard regularity constraints increase wirelength [Kong et al. 2020]. Moreover, our approach is for general 2D PE arrays, which are not necessarily systolic arrays. As such, there might be random logic cells that are preferred to be placed within the array, between PE columns and rows. This is another reason why hard regularity is difficult.

6.2 Step 0: PE Array Orientation

For a 2D PE array, its placement can be arranged to eight different orientations, all of which are regular. For example, the placement of Figure 5(a) can be rotated to (b) or flipped to (c) while its regularity is maintained. In the regularity-constrained quadratic placement of Step 1 and the array regularization in Step 2, the array orientation needs to be determined in advance. This can be realized by examining data movement and IO locations. For the example in Figure 2, the PEs that first receive elements of matrix A should be placed as the first column on the left, while the PEs that first receive elements of matrix B need to be placed as the first row on the top.

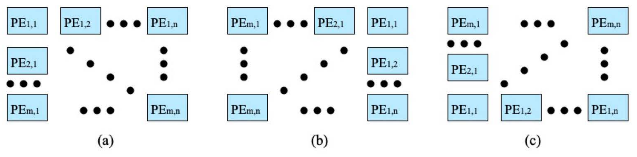


Figure 5: (a) The original orientation; (b) rotated by 90° clockwise; (c) vertical flipping.

6.3 Step 1: Initial Quadratic Placement

Step 1 in Figure 4 is an initial quadratic placement like [Lu et al. 2015] but with regularity constraints. Let $(\boldsymbol{x}, \boldsymbol{y})$ be the vectors indicating x-y coordinates of all movables in the given circuit. Quadratic placement assumes all nets are two-pin nets, while a multi-pin net can be transformed to a set of 2-pin nets using the Bound2Bound model [Kim et al. 2011]. Quadratic placement is to minimize the following quadratic wirelength without considering cell overlaps or other constraints:

$$W_Q(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{2}\boldsymbol{x}^T A_x \boldsymbol{x} + \boldsymbol{h}_x^T \boldsymbol{x} + \frac{1}{2}\boldsymbol{y}^T A_y \boldsymbol{y} + \boldsymbol{h}_y^T \boldsymbol{y} \quad (5)$$

where A_x and A_y are symmetric positive definite matrices indicating connections among movable cells, and $\boldsymbol{h}_x/\boldsymbol{h}_y$ are constant vectors implying connections with fixed cells. By making the gradient of $W_Q(\boldsymbol{x}, \boldsymbol{y})$ to be zero, the solution can be found by solving a linear system.

We apply hard regularity constraints to PE cells for the quadratic placement for three reasons. First, such a quadratic placement solution would provide the RePlAce kernel with an initial prototype with 2D regularity so that the array regularization in Step 2 becomes more accessible. Second, quadratic placement is a simple method without considering cell overlaps and thus does not conflict with the regularity constraints. Third, the regularity constraints can significantly reduce the number of variables in solving the linear system.

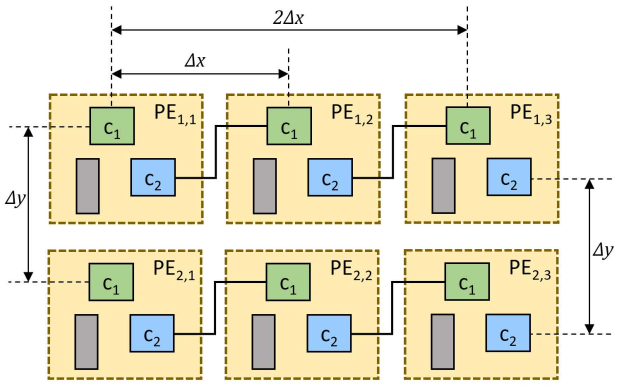


Figure 6: Hard regularity constraints for PE cells.

Consider a circuit that contains an $m \times n$ PE array, represented by $\Pi = \{PE_{1,1}, PE_{1,2}, ..., PE_{1,n}, PE_{2,1}, ..., PE_{m,n}\}$. Each $PE_{i,j} \in \Pi$ has p cells $\{c_{i,j}^1, c_{i,j}^2, ...c_{i,j}^p\}$. Identical cells of different PEs form a regularity group $G^k = \{c_{i,j}^k | i = 1, ..., m; j = 1, ..., n\}$. For the example in Figure 6, all green cells belong to G^1 and all blue cells form G^2. For the cells in the same group G^k, we enforce the following constraints

$$\begin{aligned} x_{i,j}^k &= x_{1,1}^k + (j-1) \cdot \Delta_x, \ \ i = 1, ..., m; j = 1, ..., n \\ y_{i,j}^k &= y_{1,1}^k + (i-1) \cdot \Delta_y, \ \ i = 1, ..., m; j = 1, ..., n \end{aligned} \quad (6)$$

where $(x_{i,j}^k, y_{i,j}^k)$ denote the coordinates of cell $c_{i,j}^k$, and Δ_x/Δ_y are variable column/row pitches as shown in Figure 6. The constraints here assume that we know the PE array orientation, which is determined in Step 0 of Figure 4. Even when $m = n$, the array orientation still matters.

For an $m \times n$ PE array where each PE has p cells, the original number of decision variables is $2mnp$. After enforcing constraints (6), each regularity group G^k has only 4 variables: $x_{1,1}^k, y_{1,1}^k, \Delta_x, \Delta_y$, where Δ_x and Δ_y are shared among all regularity groups. Since there are p regularity groups, the number of variables for the entire array is reduced to $2p + 2$. Such variable reduction helps speedup the quadratic placement.

6.4 Step 2: RePlAce with Array Regularization

The kernel of RePlAce is composed of iterations of Nesterov's method. For an intermediate solution v_i at the i-th iteration, the gradient of objective function is computed and applied to obtain solution v_{i+1}. To exploit the 2D regularity for a PE array, we regularize solution v_i to $\tilde{v}_i$ before the gradient computation. The RePlAce iterations with array regularization form Step 2 in Figure 4, where sub-steps are shown in the blue box on the right.

The array regularization is to take the solution v_i and force its PE array Π to be placed with 2D regularity and fit in the region occupied by the array in v_i. Let $\tilde{R}_i$ be the region occupied by Π in solution v_i, where PE placement can be irregular. The regularization changes the placement of Π cells according to regular 2D array in $\tilde{R}_i$. Please note that the area of $\tilde{R}_i$ can be smaller than the total cell area of Π in early iterations.

Algorithm 1 Array Regularization

```
Input: PE array $\Pi$ and placement solution $v_i$
Output: Regularized solution $\tilde{v}_i$, $\Pi$ placement is regular
1: $(x_{i,j}, y_{i,j}) \leftarrow$ COG of $PE_{i,j}$, $i = 1, ..., m$; $j = 1, ..., n$
2: $x_{min} \leftarrow \min_{\forall i,j} x_{i,j}$, $x_{max} \leftarrow \max_{\forall i,j} x_{i,j}$
3: $y_{min} \leftarrow \min_{\forall i,j} y_{i,j}$, $y_{max} \leftarrow \max_{\forall i,j} y_{i,j}$
4: $P_x \leftarrow (x_{max} - x_{min})/(n-1)$
5: $P_y \leftarrow (y_{max} - y_{min})/(m-1)$
6: for $i = 1 : m$ do
7:     for $j = 1 : n$ do
8:         for $k = 1 : p$ do
9:             $x^k_{i,j} \leftarrow x^k_{1,1} + (j-1)P_x - (x_{1,1} - x_{min})$
10:            $y^k_{i,j} \leftarrow y^k_{1,1} + (i-1)P_y - (y_{1,1} - y_{min})$
```

The pseudo code for the regularization is provided in Algorithm 1. It assumes that the PE array orientation is like Figure 5(c). The other orientations can be handled in the same way, and the orientation is determined in Step 0 (Section 6.2). In line 1, the COG (Center of Gravity) of each PE in solution v_i is calculated. Region $\tilde{R}_i$ is defined by the bounding box $(x_{min}, y_{min}) - (x_{max}, y_{max})$, which is obtained through lines 2 and 3. Note that the bounding box is obtained through PE COGs instead of PE cell locations. This is because PE COGs reflect the general shape of a PE array region, while PE cells may include outliers far from the PE array's mass. Lines 4 and 5 calculate the column and row pitches to be enforced in the regularization. The locations of individual PE cells are decided through lines 6-10. The first two terms on the right-hand side of lines 9 and 10 are very similar to Equation (6) except that P_x and P_y are constants while Δ_x and Δ_y are variables. The last two terms in lines 9 and 10 are to handle a special case where $PE_{1,1}$ is not at the lower-left corner of the array in v_i. This case is illustrated in Figure 7. By subtracting $x_{1,1} - x_{min}$ $(y_{1,1} - y_{min})$ in line 9 (line 10), $PE_{1,1}$ is forced to be at the lower-left corner in the regularized solution $\tilde{v}_i$.

After the array regularization, the computation of wirelength gradient $\nabla_v \tilde{W}$ in Step B.2 of Figure 4 can be accelerated due to the repeated placement patterns. One example is shown in Figure 6,

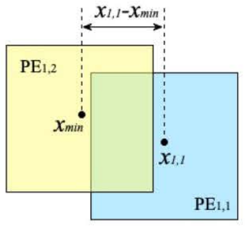


Figure 7: An example where $PE_{1,1}$ is not at the lower-left corner in solution v_i.

where the four nets in solid lines are identical. We only need to compute the gradient for one of them and then duplicate the result to the other nets of the same pattern.

The array regularization is performed only for RePlAce iterations in Step 2 of Figure 4 but not for later iterations in Step 3. There are two main reasons. First, RePlAce assumes that each cell carries a positive charge and uses an electrical repelling force to spread out cells. The resulting equipotential lines almost always have curvatures and cause uneven forces on each side of the PE array region. Hence, RePlAce tends to place a PE array into a rounded shape and conflicts with the effort of forming a rectangular array. Second, random logic cells are not placed in a regular fashion, and they have wire connections with PE cells, which tend to cause irregularity for PE cell placement. Overall, continuing the regularization throughout all RePlAce iterations easily incurs solution divergence.

7 EXPERIMENTS

7.1 Testcase Generation

To evaluate placement with 2D PE arrays, we designed three matrix multiplication circuits and six CNN (Convolutional Neural Network) circuits as test cases. Each circuit has an $m \times m$ PE array, and the overall architecture is shown in Figure 8. Besides a PE array, each circuit has data buffers and a controller that coordinates data movement on the circuit.

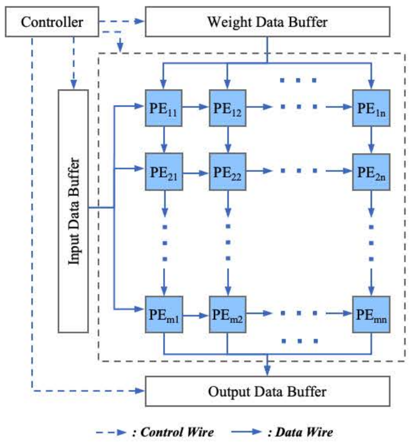


Figure 8: The overall architecture of the test cases

The six CNN circuits cover two commonly seen variants: weight stationary (WS) and output stationary (OS). Here, the weight means the filter elements for the convolution operation of CNN. In a WS design, weight data are preloaded onto the weight registers of each PE. During computation, CNN input data are fed from the left side and shifted toward the right side through the PE array, while output data move top-down and accumulate the partial sum. In an OS design, both input and weight data shift through the array while output data are accumulated locally at each PE before they are sent out. The PE designs for WS and OS cases are different and are shown in Figure 9. The controller designs for WS and OS are also different. In these designs, a PE contains near 200 cells.

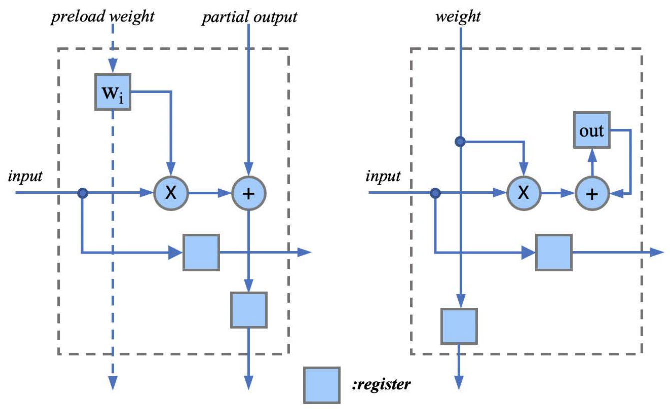


Figure 9: Weight stationary (left) and output stationary (right) CNN PE designs.

These cases are designed with 8-bit fixed-point computation, which is common for CNN circuit designs. They are synthesized by Synopsys Design Vision using Nangate 45nm cell library. The IO placement is obtained using Cadence Innovus.

Table 1: Statistics of the testcases

Testcase	#cells	#movables	#nets	% PE cells
MM 16×16	83K	82K	95K	58%
MM 32×32	326K	325K	377K	58%
MM 64×64	1292K	1290K	1497K	59%
CNNOS 16×16	83K	83K	96K	57%
CNNOS 32×32	326K	325K	378K	58%
CNNOS 64×64	1293K	1291K	1498K	58%
CNNWS 16×16	83K	82K	95K	59%
CNNWS 32×32	328K	327K	379K	60%
CNNWS 64×64	1306K	1305K	1511K	60%

The major characteristics of the test cases are summarized in Table 1, where MM means matrix multiplication, CNNOS indicates output stationary design of CNN and CNNWS denotes weight stationary design of CNN. The PE array sizes are also provided in the first column. Each large case has near 1.3 million cells and near 1.5 million nets. The number of PE cells accounts for 56% – 60% of the total number of cells.

7.2 Experiment Setup

The experiments were performed on a computer with AMD 3700X processor of 4.05GHz and 16GB memory and 64-bit Linux OS. Our approach is compared with ReRlAce [Cheng et al. 2019], NTU-place3 [Chen et al. 2008] and POLAR [Lin et al. 2013]. The implementation of our approach is based on the source code of RePlAce. The results of RePlAce[1] and NTUplace3[2] are obtained through running downloaded software. We implemented POLAR and matched its results on ISPD2005 contest benchmark. The global placement solutions of these methods are continued with the detailed placement of NTUplace3, where the target utilization is 1.0. Steiner tree wirelength is estimated using FLUTE [Chu and Wong 2008].

7.3 Main Results

The main results are summarized in Table 2. On average, our approach reduces HPWL by 5.97%, 11.8%, 13.3% over RePlAce, NTU-pLACE3, and POLAR, respectively. In addition, our approach results in 12% less Steiner tree wirelength than RePlAce. A close look tells that our Steiner tree wirelength results are usually similar to HPWL, while RePlAce results in a much more significant difference. This observation is consistent with that in datapath layout [Ward et al. 2013]. Although our approach increases CPU runtime, the increase is limited. For instance, the runtime difference from NTUplace is less than 5%. We set the timeout limit as 3 hours, which is about 6× the longest runtime among all cases. NTUplace3 cannot complete CNNWS 64×64 within this limit.

7.4 Other Results

The effect of hard regularity constraints in initial quadratic placement (Section 6.3) is examined, and the results are shown in Table 3. Although the regularity constraints increase HPWL at the quadratic placement stage, they contribute to the overall HPWL reduction after detailed placement. Moreover, the constraints reduce quadratic placement runtime by 38.9%.

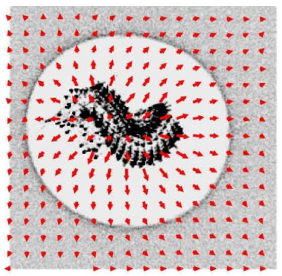


Figure 10: Electrical fields and an equipotential line for an intermediate RePlAce solution on CNNOS 16×16.

In Section 6.4, it is mentioned that the array regularization cannot continue throughout all RePlAce iterations as the rectangle regularity would conflict with the tendency of curving equipotential lines in RePlAce. In Figure 10, the electrical fields of an intermediate RePlAce solution are indicated by short red arrows, and the circle is an equipotential line. We further investigate the effect of increasing/decreasing the number of iterations with array regularization. Suppose Step 2 of Figure 4 has I_2 iterations, where the array regularization is performed. If $I_2 = 0$, our approach is reduced to the original RePlAce. We vary the value of I_2 and observe the effect on solution convergence and HPWL. This experiment was conducted for CNNOS 16×16, with two variants where the target overflows in

[1]https://github.com/The-OpenROAD-Project/RePlAce/tree/standalone
[2]https://github.com/The-OpenROAD-Project/RePlAce/tree/standalone/ntuplace

Table 2: Main results on HPWL ($\times 10^9$), Steiner tree wirelength stWL ($\times 10^9$) and CPU runtime (s).

	RePlAce			NTUplace3		POLAR		Ours		
Testcase	HPWL	stWL	CPU	HPWL	CPU	HPWL	CPU	HPWL	stWL	CPU
MM 16× 16	5.89	6.34	38	6.07	37	6.14	33	5.76	5.98	60
MM 32× 32	23.49	25.84	280	24.77	281	25.17	188	22.92	23.49	494
MM 64× 64	96.45	104.23	1406	105.25	1063	107.04	968	86.31	89.85	1827
CNNOS 16×16	5.89	6.60	39	6.35	40	6.22	36	5.71	6.11	37
CNNOS 32×32	26.70	29.89	261	28.16	291	28.59	229	24.85	26.83	151
CNNOS 64×64	99.25	109.84	1537	107.71	1120	109.80	929	89.02	92.31	1681
CNNWS 16×16	5.92	6.37	41	6.3	41	5.94	31	5.80	5.90	43
CNNWS 32×32	26.56	28.71	203	26.98	328	27.52	225	25.49	25.40	256
CNNWS 64×64	99.32	107.32	1195	N/A	Timeout	111.16	915	91.27	91.55	1406
Norm average	1.059	1.120	0.918	1.118	0.958	1.133	0.750	1.000	1.000	1.000

Table 3: Effects of hard regularity constraints in quadratic placement. QP: Quadratic Placement. DP: Detailed Placement.

	QP with regularity constraints				QP without regularity constraints			
Testcase	QP HPWL	QP CPU(s)	DP HPWL($\times 10^9$)	Total CPU(s)	QP HPWL	QP CPU(s)	DP HPWL($\times 10^9$)	Total CPU(s)
CNNWS 16×16	32403	2.03	5.80	43	27067	2.72	5.91	47
CNNWS 32×32	87249	7.78	25.49	256	85818	11.19	25.60	277
CNNWS 64×64	352790	35.91	91.27	1406	328115	49.95	95.58	1772
Norm average	1.000	1.000	1.000	1.000	0.91	1.389	1.018	1.145

global placement were set to 0.1 and 0.2, respectively. The convergence results are plotted in Figure 11 where the dashed line means that there is no convergence. The leftmost dots correspond to the original RePlAce. One can see that the solution convergence becomes difficult when I_2 is large. Especially when the target overflow is 0.1, there is no convergence as I_2 reaches 250. The post-detailed placement HPWL results are shown in Figure 12. There is a sweet spot for HPWL minimization, and HPWL increases when I_2 is too large. These results confirm that the array regularization cannot be carried out throughout all RePlAce iterations.

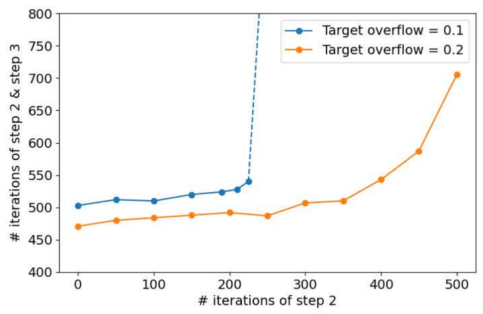


Figure 11: Effect of increasing #iterations of Step 2.

In Step B.2 of Figure 4, the computation of wirelength function gradient $\nabla_v \tilde{W}$ is accelerated by making use of the regularity. Table 4 shows the effect of this technique. Compared to computations without this acceleration, this technique can reduce the global placement runtime by about 30%.

7.5 Placement Demonstration

In order to obtain an intuition of the soft regularity in our approach, we plot placement visualizations for CNNWS 16×16 in Figure 13,

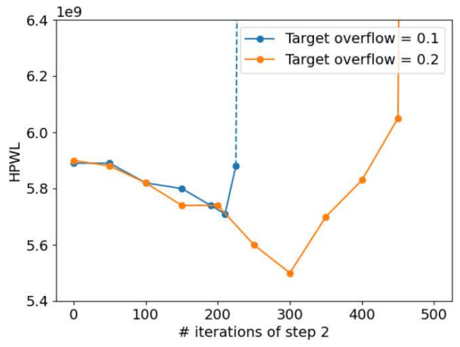
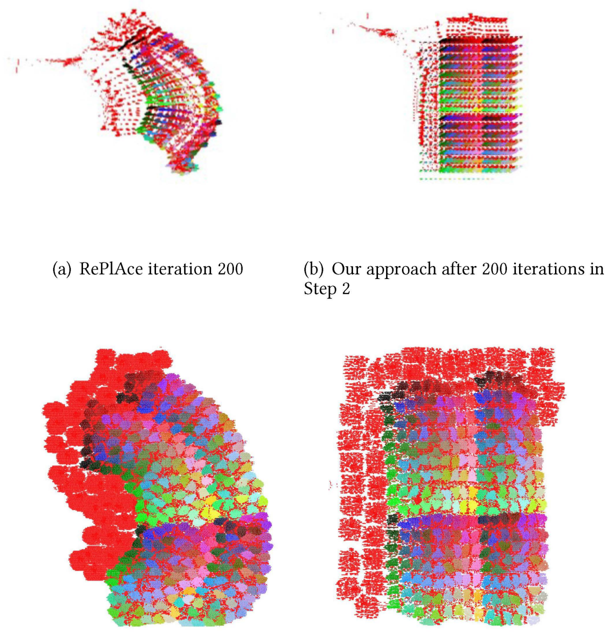


Figure 12: HPWL vs. #iterations of Step 2.

Table 4: Global placement runtime (in seconds) comparison between our approach with Step B.2 in Figure 4, where the computation of $\nabla_v \tilde{W}$ is accelerated, and ordinary computation without the acceleration.

Testcase	Fast $\nabla_v \tilde{W}$	Ordinary $\nabla_v \tilde{W}$
CNNOS 16×16	37	43
CNNOS 32×32	151	196
CNNOS 64×64	1681	2427
Average	1.00	1.30

where cells of the same PE share the same color and all random logic cells are in red color. In (a) and (b), we compare intermediate solutions from RePlAce and our approach at the same iteration. In our approach, the first 200 iterations are Step 2 in Figure 4 where array regularization is performed. The placement regularity of the PE array is evident in Figure 13(b), while the regularity in Figure 13(a)



(a) RePlAce iteration 200

(b) Our approach after 200 iterations in Step 2

(c) Detailed placement from RePlAce

(d) Detailed placement from our approach

Figure 13: (a) Placement of RePlAce at iteration 200; (b) Placement after 200 iterations in Step 2 of our approach; (c) Final detailed placement resulted from RePlAce; (d) Final detailed placement resulted from our approach.

is much weaker. The final solutions after detailed placement are compared in Figures 13(c) and (d). One can see that the PE array from our approach shows an approximated regularity or soft regularity while RePlAce tends to form a rounded shape.

8 CONCLUSION AND FUTURE WORK

The regularity of 2D PE arrays allows placement tools to improve for a category of designs with growing popularity, such as CNN circuits. This work proposes a global placement technique for exploiting the regularity based on RePlAce, the latest state-of-the-art placement framework. Experimental results show that the proposed technique can reduce half perimeter wirelength and Steiner tree wirelength by 6% and 12%, respectively. In future research, we will extend this technique to handle routability.

ACKNOWLEDGMENTS

This work is partially supported by NSF CCF-1937396, CMMI-2038625, NSF CCF-1854737 and CCF-2106725.

REFERENCES

Takahiro Asai and Tadashi Matsumoto. 2000. A systolic array RLS processor. In *IEEE Vehicular Technology Conference.* 2247–2251.

H Cai, Stefaan Note, Paul Six, and Hugo De Man. 1990. A data path layout assembler for high performance DSP circuits. In *ACM/IEEE Design Automation Conference.* 306–311.

Long-Wen Chan and Ming-Young Chen. 1988. A new systolic array for discrete Fourier transform. *IEEE Transactions on Acoustics, Speech, and Signal Processing* 36, 10 (1988), 1665–1666.

Tung-Chieh Chen, Zhe-Wei Jiang, Tien-Chang Hsu, Hsin-Chen Chen, and Yao-Wen Chang. 2008. NTUplace3: An analytical placer for large-scale mixed-size designs with preplaced blocks and density constraints. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 27, 7 (2008), 1228–1240.

Yu-Hsin Chen, Tien-Ju Yang, Joel Emer, and Vivienne Sze. 2019. Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 9, 2 (2019), 292–308.

Chung-Kuan Cheng, Andrew B Kahng, Ilgweon Kang, and Lutong Wang. 2019. RePlAce: Advancing solution quality and routability validation in global placement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 38, 9 (2019), 1717–1730.

Sheng Chou, Meng-Kai Hsu, and Yao-Wen Chang. 2012. Structure-aware placement for datapath-intensive circuit designs. In *ACM/IEEE Design Automation Conference.* 762–767.

Chris Chu and Yiu-Chung Wong. 2008. FLUTE: Fast Lookup Table Based Rectilinear Steiner Minimal Tree Algorithm for VLSI Design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 27, 1 (2008), 70–83. https://doi.org/10.1109/TCAD.2007.907068

Zhuolun He, Peiyu Liao, Siting Liu, Yuzhe Ma, Yibo Lin, and Bei Yu. 2021. Physical synthesis for advanced neural network processors. In *ACM/IEEE Asia and South Pacific Design Automation Conference.* 833–840.

Atef Ibrahim, Hamed Elsimary, and Fayez Gebali. 2018. New systolic array architecture for finite field division. *IEICE Electronics Express* 15, 11 (2018), 1–11.

Norman P. Jouppi et al. 2017. In-datacenter performance analysis of a tensor processing unit. In *ACM International Symposium on Computer Architecture.* 1–12.

Myung-Chul Kim, Dong-Jin Lee, and Igor L Markov. 2011. SimPL: An effective placement algorithm. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 31, 1 (2011), 50–60.

Hongxin Kong, Lang Feng, Chunhua Deng, Bo Yuan, and Jiang Hu. 2020. How Much Does Regularity Help FPGA Placement?. In *IEEE International Conference on Field-Programmable Technology.* 76–84.

Tao Lin, Chris Chu, Joseph R Shinnerl, Ismail Bustany, and Ivailo Nedelchev. 2013. POLAR: Placement based on novel rough legalization and refinement. In *IEEE/ACM International Conference on Computer-Aided Design.* 357–362.

Jingwei Lu, Hao Zhuang, Pengwen Chen, Hongliang Chang, Chin-Chih Chang, Yiu-Chung Wong, Lu Sha, Dennis Huang, Yufeng Luo, Chin-Chi Teng, and Chung-Kuan Cheng. 2015. ePlace-MS: Electrostatics-based placement for mixed-size circuits. *IEEE Transactions on Computer-Aided Design* 34, 5 (2015), 685–698.

Raymond XT Nijssen and Jochen AG Jess. 1996. Two-dimensional datapath regularity extraction. In *IFIP Workshop on Logic and Architecture Synthesis.* 110–117.

Johan Peltenburg, Shanshan Ren, and Zaid Al-Ars. 2016. Maximizing systolic array efficiency to accelerate the PairHMM forward algorithm. In *IEEE International Conference on Bioinformatics and Biomedicine.* 758–762.

Patrice Quinton. 1987. An introduction to systolic architectures. In *Future Parallel Computers.* Springer Berlin Heidelberg, 387–400.

Mahendra Vucha and Arvind Rajawat. 2011. Design and FPGA implementation of systolic array architecture for matrix multiplication. *International Journal of Computer Applications* 26, 3 (2011), 18–22.

Samuel Ward, Duo Ding, and David Z. Pan. 2012. PADE: A High-Performance Mixed-Size Placer with Automatic Datapath Extraction and Evaluation through High-Dimensional Data Learning. In *ACM/IEEE Design Automation Conference.* 756–761. https://doi.org/10.1145/2228360.2228497

Samuel I Ward, Myung-Chul Kim, Natarajan Viswanathan, Zhuo Li, Charles J Alpert, Earl E Swartzlander, and David Z Pan. 2013. Structure-aware placement techniques for designs with datapaths. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 32, 2 (2013), 228–241.

Xuechao Wei, Cody Hao Yu, Peng Zhang, Youxiang Chen, Yuxin Wang, Han Hu, Yun Liang, and Jason Cong. 2017. Automated systolic array architecture synthesis for high throughput CNN inference on FPGAs. In *ACM/IEEE Design Automation Conference.* 1–6.

Changqi Yang, Xianlong Hong, Yici Cai, Wenting Hou, Tong Jing, and Weimin Wu. 2003. Standard-cell based data-path placement utilizing regularity. In *IEEE International Conference on ASIC.* 97–100.

T Tao Ye and Giovanni De Micheli. 2000. Data path placement with regularity. In *IEEE/ACM International Conference on Computer-Aided Design.* 264–270.

Jiaxi Zhang, Wentai Zhang, Guojie Luo, Xuechao Wei, Yun Liang, and Jason Cong. 2019. Frequency improvement of systolic array-based CNNs on FPGAs. In *IEEE International Symposium on Circuits and Systems.* 1–4.