Systolic Array Placement on FPGAs

Hailiang Hu*, Donghao Fang*, Wuxi Li†, Bo Yuan‡, Jiang Hu*§
*Dept. of Electrical and Computer Engineering, Texas A&M University
§Dept. of Computer Science and Engineering, Texas A&M University
†AMD, Inc. ‡Dept. of Electrical and Computer Engineering, Rutgers University
{hailiang, donghao, jianghu}@tamu.edu; wuxi.li@amd.com; bo.yuan@soe.rutgers.edu

2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD) | 979-8-3503-2225-5/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICCAD57390.2023.10323742

***Abstract*—Systolic array designs have regained popularity in recent years, particularly for their applications in accelerating CNN (Convolutional Neural Network) computing in hardware, including on FPGAs. However, existing FPGA layout techniques are primarily designed for general-purpose applications and have not fully leveraged the regularity of systolic arrays to enhance solution quality. This paper presents a new algorithmic approach for systolic array placement on FPGAs. Our approach enables $23\% - 25\%$ wirelength reduction for CNN circuits compared to an industrial tool and state-of-the-art academic methods. Moreover, it usually leads to significantly reduced routing resource utilization, accelerated placement runtime and improved timing performance.**

***Index Terms*—Systolic Arrays, Placement, FPGA, CNN**

I. Introduction

Systolic array is a VLSI architecture [1] that enables massively parallel datapath computing with a highly regular topology and local data interconnects. In recent years, systolic arrays have regained popularity due to their applications in hardware acceleration for CNN (Convolutional Neural Network) computing, such as in Google TPU [2] and FPGA-based CNN computing acceleration [3]. However, the regular topology of systolic arrays has rarely been leveraged in automatic layout tools. In a study [4], it was noted that an FPGA placement tool overlooked regularity in a systolic array-based CNN design and the circuit timing could be significantly enhanced by incorporating regularity-based constraints. The 2D regularity of systolic arrays was exploited in ASIC (Application-Specific Integrated Circuit) cell placement [5] and facilitated significant wirelength reduction. The regularity was also utilized to largely accelerate simulated annealing-based FPGA placement [6].

In this work, we study systolic array placement on FPGAs. In particular, we consider 2D systolic arrays, which are the most typical topology for CNN computing. Compared to ASICs, systolic array placement on FPGAs is notably more challenging. Since an ASIC silicon die is generally an open space for placement, a systolic array can be placed onto it with little restriction. By contrast, an FPGA consists of pre-fabricated elements, such as CLBs (Configurable Logic Blocks), DSP blocks and RAM blocks, which impose significant hard constraints on the placement of a systolic array. A typical FPGA architecture is illustrated in Figure 1. The heterogeneity resulting from the CLB/DSP/RAM columns makes it very difficult to directly utilize the regularity of a systolic array, except in a few special cases.

Typically, a systolic array consists of a 2D array of Processing Elements (PEs), where each PE executes Multiply and Accumulate (MAC) operations. When placing systolic arrays on FPGAs, the previous work [6] chose to place the PEs onto CLBs, which are less efficient for datapath computing compared to DSP blocks. Indeed, placing systolic array PEs on DSPs is more prevalent [4], [7]. However, DSP columns are normally sparser than CLB columns, making it more challenging to accommodate a regular array of PEs.

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

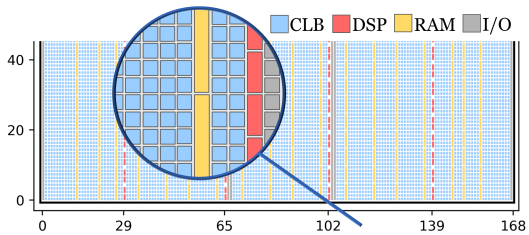


Figure 1: A typical FPGA architecture.

Systolic arrays are placed on DSPs in RapidLayout [7] using off-the-shelf evolutionary algorithm packages without dedicated attention to the 2D regularity. Also, RapidLayout is geared to solve a special problem of placing convolutional units, each of which occupies 18 DSPs, rather than typical systolic array placement [4]–[6], where each PE corresponds to one DSP block. In [4], a greedy floorplan constraint is enforced for an existing FPGA layout tool to improve regularity of systolic array placement. State-of-the-art FPGA placement techniques [8]–[11] are general-purpose and do not distinguish systolic arrays from other designs. They also simultaneously handle CLBs, DSPs and RAMs. Overall, research on systolic array placement on FPGAs is quite limited and there lacks an efficient approach for leveraging the regularity of general CNN circuit designs.

In this work, we introduce a new approach to 2D systolic array placement on FPGAs exploiting the regularity. This approach is centered around a Region-wise Sweep in Alternating Direction (R-SAD) algorithm for placement on a single DSP column, which can simultaneously minimize intra-column wirelength and facilitate minimum inter-column wirelength. R-SAD achieves near-optimal solutions compared to ILP (Integer Linear Programming) solving while being tremendously faster. We also propose a partition enumeration and pruning technique, which is integrated with R-SAD for generating 2D array placement solutions. When used as a pre-processing for general FPGA placement, our approach usually leads to improvement on all of wirelength, routing resource utilization, placement runtime and circuit timing performance for CNN designs. The main contributions of this work are summarized as follows.

- We develop a linear complexity Region-wise Sweep in Alternating Direction (R-SAD) algorithm for placement of systolic array kernels on a single DSP column. R-SAD achieves near-optimal solution compared to ILP solving and is over $50K\times$ faster.
- A partition enumeration and pruning technique is developed to utilize R-SAD for 2D array placement, which is integrated with general FPGA placement for the layout of systolic-array based CNN circuit designs.
- We demonstrate an improvement of 23% - 25% in half-perimeter wirelength, 19% - 22% in routing resource utilization, and 3% to 10% less runtime compared to existing industrial and academic FPGA placement tools.

- Our approach increases the maximum frequency by 1% -3% on average compared to an existing industrial and academic FPGA placement tools.

II. Systolic Array-Based CNN Circuit

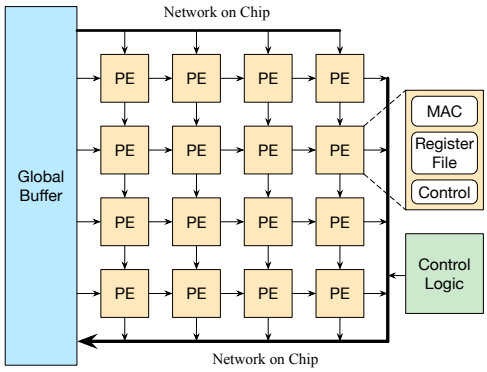


Figure 2: A systolic array-based CNN circuit architecture.

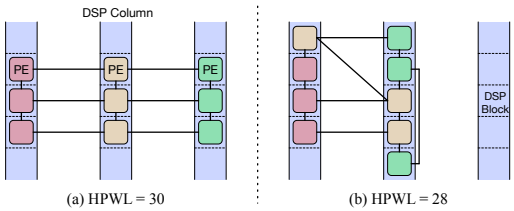
A typical architecture for a systolic array-based CNN circuit can be seen in Figure 2. The architecture includes a global buffer, often implemented using SRAM, a 2D array of PEs (Processing Elements), and control logic beside the PE array. Data communication is usually handled using a network-on-chip. Each PE consists of a MAC (Multiply and Accumulate) unit, a register file, and intra-PE control circuitry. The systolic array, which is the 2D PE array, not only has a regular topology but also has only local data interconnects. This means that the data interconnects between PEs are restricted to their neighbors.

III. How to Leverage the Regularity?

A straightforward approach to leverage the regularity is to retain the regularity as the 3×3 array example shown in Figure 3(a). However, such an approach has some significant drawbacks:

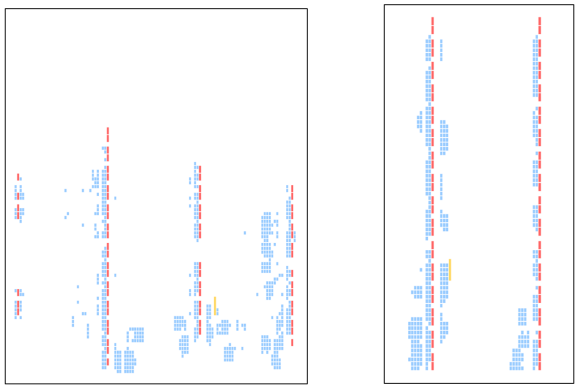
1) It is often infeasible. The number of DSP columns in an FPGA is usually less than 10 while the systolic array dimensions of a CNN circuit can easily reach 16×16 and beyond. Hence, it is often impossible to let each systolic array column to occupy an entire DSP column for keeping the regularity.
2) Even when the number of systolic array columns is no greater than the number of DSP columns, this approach may result in inferior solutions due to the fact that the distance between two neighboring DSP columns is often much larger than the distance between two neighboring DSP blocks in the same column. For the example in Figure 3, if we pack the three systolic array columns into two DSP columns as in (b), the HPWL (Half-Perimeter Wire-Length) can be reduced from 30 to 28, although the regularity is not retained.
3) When keeping the regularity is infeasible, even an effort to approximate the regularity can be harmful for the same reason as the previous item. Figure 4 demonstrates the placement solutions for a CNN circuit with an 8×8 systolic array from Vivado and our placement method. The Vivado solution occupies 4 DSP columns and is closer to the square aspect ratio of $1:1$ compared to our approach. However, the solution of our method reduces HPWL by 13%.

Without an effort to retain or approximate the regularity, how can we leverage the regularity for better placement solutions? Here is the answer: the regularity provides a nice problem structure that facilitates efficient solution search for minimizing wirelength. An analogy is that it is much easier to find the maximum value in a set of sorted numbers than in a set of unsorted numbers. Performing evolutionary algorithms for the placement of a regular structure can be likened to searching for the maximum value among sorted numbers using enumeration. Indeed, the regularity of systolic arrays is heavily utilized in our R-SAD algorithm and partition enumeration/pruning technique.



(a) HPWL = 30 (b) HPWL = 28

Figure 3: Placement of 3×3 systolic array.



(a) Vivado placement, $HPWL = 2.47 \times 10^4$ (b) Our systolic array placement, $HPWL = 2.15 \times 10^4$

Figure 4: Impact of systolic array placement. Red color indicates PEs placed on DSPs, and light blue implies circuits placed on CLBs.

IV. CNN Circuit Placement Methodology

Usually, a DSP block can only accommodate the MAC unit of a PE and the rest of the PE needs to be placed in CLB/RAMs nearby. As such, we propose the following two-phase methodology for the placement of systolic array-based CNN circuits.

- Phase 1: MAC array placement on DSPs by our algorithm.
- Phase 2: placement of the rest circuit, including global buffer, control logic, and register files and intra-PE control of PEs, using a conventional FPGA tool with the MAC array locations fixed.

One can treat the MACs as the backbone of a systolic array. By considering only the backbone, Phase 1 can be focused on leveraging the regularity without worrying other details, such as the connections between register files and MACs. Once the locations of MACs are fixed after Phase 1, the other wirelength, e.g., those between register files and MACs, and the connections between PEs and the global buffers, can be minimized by conventional tools in Phase 2, which are the best techniques so far for handling irregular designs. Please note that there is no direct wire connection between IO pins and PEs as IO pins are mostly connected with the global buffer. As such, the wirelength related to IO pins is also minmized in Phase 2.

V. MAC Array Placement Problem

Given an $m \times n$ array of MACs, each of which corresponds to a PE in the given systolic array, and a set of DSP columns of a specific FPGA architecture, our algorithm is to place the MACs onto DSP

slots, where DSP blocks are located, such that the total wirelength among the MACs is minimized.

The $m \times n$ MAC array can be described by a grid graph $G=(V, E)$, where $V=\{v_1, v_2, ..., v_{|V|}\}$ is a set of vertices, each corresponding to a MAC, and $E=\{e_1, e_2, ..., e_{|E|}\}$ is a set of edges indicating the interconnect between the PEs corresponding to the MACs. Please note $|V|=m \cdot n$ and $|E|=m(n-1)+n(m-1)$. A vertex $v \in V$ can be alternatively represented by $v_{i,j}$, where i and j are row and column indices, respectively, and $1 \leq i \leq m$ and $1 \leq j \leq n$. The edge connecting two vertices $v_{i,j}$ and $v_{p,q}$ can be represented as $e(v_{i,j}, v_{p,q})$. The edges are restricted to neighboring vertices, i.e., there is an edge between $v_{i,j}$ and $v_{p,q}$ if and only if $|i-p|+|j-q|=1$.

The given DSP slots also form a $k \times l$ array and can be represented by $P = \{p_{1,1}, p_{2,1}, ..., p_{k,1}, p_{1,2}, p_{2,2} ..., p_{k,l}\}$, where $p_{i,j}$ indicates a slot in the i-th row and j-th column. Each slot $p_{i,j} \in P$ has a location specified by $x(p_{i,j}) = j \cdot d_H$ and $y(p_{i,j}) = i \cdot d_V$, where d_H (d_V) is the horizontal (vertical) distance between adjacent DSP columns (rows), respectively. Normally, $d_H >> d_V$, $k >> l$ and $n > l$.

MAP (MAC Array Placement) Problem: Given a grid graph $G = (V, E)$ representing a MAC array and an array of DSP slots P satisfying $|P| \geq |V|$, find the mapping $f : V \rightarrow P$ such that the total half-perimeter wirelength (HPWL)

$$L(f) = \sum_{e(v_i, v_j) \in E} |x(f(v_i)) - x(f(v_j))| + |y(f(v_i)) - y(f(v_j))|$$

is minimized.

VI. MAC Array Placement Algorithm

A. Overview

Our MAC array placement algorithm consists of three steps:

- Step 1: Partition candidate generation. The given MAC array is partitioned into sub-arrays, and each sub-array is to be placed onto a distinct DSP column. Multiple partitioning candidate solutions are generated through enumeration.
- Step 2: Partition candidate pruning. By estimating the upper and lower bound of wirelength, the candidates that would lead to inferior placement solutions are pruned out.
- Step 3: Placement of remaining candidates. Placement is performed for the remaining candidates using our R-SAD (Region-wise Sweep in Alternating Directions) algorithm and the solution with the minimum wirelength is selected.

B. Step 1: Partition Candidate Generation

In this step, the given MAC array is partitioned into a set of sub-arrays, each of which will be placed onto a distinct DSP column. For example, in Figure 5(b), the 5×3 array is partitioned into two sub-arrays, which are indicated by different colors and will be placed onto two separate DSP columns. At this step, it is unclear which partitioning would eventually lead to the solution with the minimum HPWL. Thus, we enumerate multiple candidate solutions, including the case of single sub-array like in Figure 5(a).

We preserve the MAC columns intact during the partitioning, i.e., the MACs in the same column are never partitioned into different sub-arrays. There are two motivations behind this approach. First, the dimension of a MAC array is usually close to square ($m \approx n$) whereas the DSP arrays are generally tall and thin ($k >> l$). Therefore, a MAC column height is less than that of a DSP column ($m < k$) when $|P| \geq |V|$. Therefore, we can usually pack one or multiple MAC columns into a single DSP column and there is no need to split a MAC column into different DSP columns. Second, with MAC columns intact, we can enumerate partitioning options in an exhaustive manner with polynomial complexity.

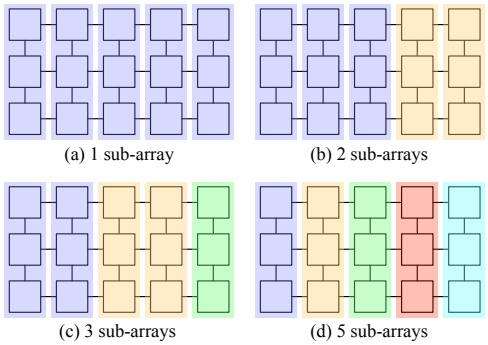


Figure 5: Enumerating different partitioning options.

Algorithm 1 Partition candidate generation

Input: MAC array columns $\{q_1, q_2, ..., q_n\}$, # DSP columns l
Output: Set of partition candidates C

```
1:  C ← ∅                                          ▷ Set of candidates
2:  for i = 1 to l do                              ▷ i: # sub-arrays in a candidate
3:      c ← ∅                                      ▷ Set of sub-arrays as a candidate
4:      w = ⌈n/i⌉                                  ▷ Max # columns per sub-array
5:      j = 1                                      ▷ Index for sweeping MAC columns
6:      while j ≤ n do
7:          if j + w ≤ n then
8:              s ← {q_j, q_{j+1}, ..., q_{j+w-1}}   ▷ s is a sub-array
9:          else
10:             s ← {q_j, q_{j+1}, ..., q_n}
11:         c ← c ∪ {s}; j = j + w
12:     C ← C ∪ {c}
13: return C
```

The pseudo-code of our partition candidate generation is provided in Algorithm 1. The set of candidates is initialized in line 1. The for-loop starting from line 2 enumerates candidates of different sizes in terms of the number of sub-arrays. Figure 5 shows candidates with sizes of 1, 2, 3 and 5. For each size, we obtain at most one candidate, which is a set of sub-arrays initialized in line 3. Line 4 defines the width or the maximum number of columns for the sub-arrays in a candidate. For example, in Figure 5(c), $w = \lceil 5/3 \rceil = 2$. The while-loop starting from line 6 sweeps the MAC columns from left to right, and j is the MAC column index for this sweeping. In each iteration (lines 7 and 8) except the last one, w MAC columns starting from column j are taken to form a sub-array. In the last iteration (line 10), the remaining MAC columns form a sub-array. These sub-arrays are added into candidate c in line 11 and the generated candidate c is added to C in line 12. Sometimes, no candidate is obtained for a certain size i. For the example in Figure 5, when i=4, w=2 and the algorithm ends up with the candidate with 3 sub-arrays as in (c).

Although the algorithm appears to be simple, it is carefully designed to achieve the following important properties.

- Property 1. Each candidate has sub-arrays of at most two different widths, one is w and the other is smaller than w. Since the placement solutions of sub-arrays with the same width are identical, the small number of sub-array widths avoids the runtime of generating many different sub-array placement solutions.

- Property 2. For each candidate, at most one sub-array has a width smaller than w. For the example in Figure 5, we do not allow partitioning $\{\{q_1, q_2\}, \{q_3\}, \{q_4\}, \{q_5\}\}$, where $w = 2$. This is why there is no 4-sub-array candidate in Figure 5. This property is required by the candidate pruning in Step 2.
- Property 3. The sub-array with width smaller than w cannot be sandwiched between two sub-arrays with size w. For example, in Figure 5(c), the green sub-array is not in the middle between two other sub-arrays. This property will help our R-SAD-based placement to minimize wirelength.

These properties can be achieved by our simple algorithm partly due to the regularity in the problem. This is a place where the regularity is exploited for efficient solution search.

C. Step 2: Partition Candidate Pruning

In this step, some partition candidates are pruned out, and only the remaining ones are forwarded to Step 3 for placement in order to reduce computation runtime. The pruning is based on estimated bounds on post-placement HPWL, and only those candidates leading to inferior placement solutions are pruned. An upper bound and a lower bound for placed HPWL is obtained for each candidate. The pruning criterion is that *a candidate is pruned out if its HPWL lower bound is greater than the upper bound of any other candidate.* As such, the pruning process is fairly straightforward once the bounds are established and the bound estimation is the key.

The bound estimate is composed by two parts: *intra-column* HPWL and *inter-column* HPWL. Here, the column means DSP columns, each of which accommodates one sub-array of a partition candidate. According to our R-SAD placement in Step 3 (Section VI-D), the **estimated lower bound for intra-column HPWL** for placing an $m \times h$ sub-array is given by

$$L_I = \min_g [-\frac{2}{3}g^3 + 2hg^2 + (\frac{2}{3} - h^2 - h)g + mh^2 + mh - m - h] \quad (1)$$

where g is an integer in $[1, \frac{\min(m,h)}{2}]$. The value of g for the minimum on the RHS can be found through linear search. The derivation of this estimate is given in Section VI-D. This bound is based on the observation that R-SAD always achieves the same or better solutions than an ILP solver, but does not have theoretic guarantee.

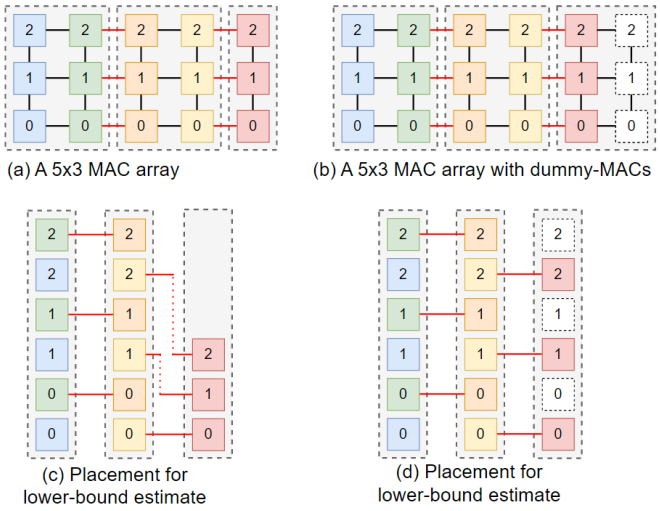


Figure 6: Placement for HPWL upper-bound, lower-bound estimate.

1) HPWL Lower Bound:

- Intra-column. The intra-column part of the lower bound is given by Equation (1). Please note the L_I for the rightmost sub-array or DSP column can be different from the other sub-arrays (DSP columns) as shown in Figure 6(c). In the intra-column lower bound estimation, we assume that all MACs of the same DSP column are placed in a contiguous manner without any vacant slots among them. Evidently, placing MACs with vacant slots in the middle increases the intra-column HPWL.
- Inter-column. The inter-column wires include horizontal and vertical segments, as shown by the red lines in Figure 6. In the lower bound estimate, the vertical segments are ignored. The horizontal inter-column wirelength is the product between d_H, the horizontal distance between two neighboring DSP columns, and the number of inter-column nets. In Figure 6(c), this value is $6d_H$.

2) HPWL Upper Bound: Please note this is an upper bound for the minimum HPWL placement solution as the upper bound for arbitrary solutions is infinite. The upper bound estimate starts with a dummy MAC padding process. If the rightmost sub-array or DSP column has less MAC columns than the other sub-arrays, dummy MAC columns are padded into this sub-array so that all DSP columns have the same number of MACs. The padding is illustrated in Figure 6(b), where the white boxes with dotted boundaries indicate dummy MACs. This is where the second property of Algorithm 1 is utilized. Then, the bound estimate assumes placement including the dummy MACs.

- Intra-column. The intra-column part of the upper bound is also given by Equation (1). However, the resulting HPWL is greater than or equal to the lower bound due to the padding of dummy MACs.
- Inter-column. When all DSP columns have the same number of MACs, including dummy MACs, our R-SAD-based placement can guarantee that only horizontal wires are needed between two neighboring columns. This is demonstrated in Figure 6(d). Therefore, the inter-column part of the upper bound is equal to the inter-column part of the lower bound estimate.

D. Step 3: Placement of Partition Candidates Using R-SAD

A candidate is a MAC array that is partitioned into a set of sub-arrays, each of which is to be placed onto a DSP column. The remaining partition candidates after the pruning in Step 2 are placed, and the minimum wirelength one among the candidates is selected to be the final solution. The key part of this step is the R-SAD (Region-wise Sweep in Alternating Directions) algorithm for placing a sub-array on a single DSP column to minimize intra-column wirelength in linear time. Due to the regularity of MAC arrays, R-SAD can be applied for all sub-arrays in a way such that the inter-column wirelength is also minimized.

Placing a sub-array onto a DSP column to minimize intra-column wirelength is very similar to the traditional linear placement problem [12], which is known to be NP-hard. However, a sub-array has a very regular topology. We exploit the regularity to derive some useful constraints and obtain the R-SAD algorithm.

1) Constraints Used in R-SAD: Due to the regularity, there could be redundant solutions with the same wirelength. We enforce the following constraints to reduce solution space without missing the optimal solution.

Constraint 1: The MAC $v_{1,1}$ in the lower-left corner of a sub-array is always placed at the lowest slot of the DSP column.

Constraint 2 ***Order Consistency (OC)*****:** If MAC u is to the lower-left of MAC v in the sub-array, then MAC u must be placed in a lower slot than MAC v in the DSP column.

$$(i_u \leq i_v) \wedge (j_u \leq j_v) \implies y(f(u)) \leq y(f(v)),$$

where i_u, i_v, j_u and j_v denote the row and column indices of MACs u and v, respectively. This constraint effectively reduces the solution space and simplifies the wirelength calculation in Section VI-D2.

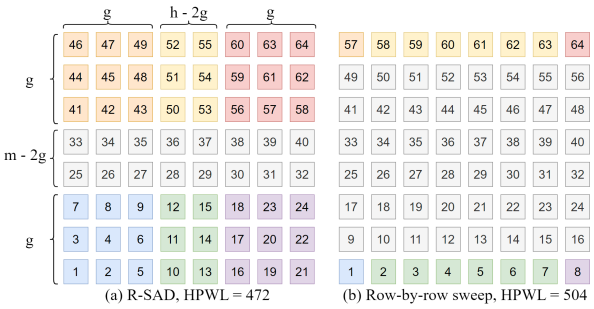


Figure 7: Placement of an 8 x 8 MAC sub-array. The number in each MAC denotes its y-coordinate after placement. Each color denotes a region.

2) **R-SAD** *(Region-wise Sweep in Alternating Directions):* In order to place an $m \times h$ sub-array onto a DSP column with the minimum wirelength, a naïve approach is row-by-row sweep. More specifically, the first row at the bottom of a sub-array is first placed at the bottom of the DSP column, then the 2nd row is placed right above, and so on. For example, row-by-row sweep for the 3×2 sub-arrays in Figure 6(a) leads to the placement in DSP columns in 6(c), which has the minimum wirelength. However, such a naïve approach results in inferior solutions when the sub-array size becomes large. In Figure 7, where the numbers in squares indicate y-coordinates in the DSP column, the row-by-row sweep solution has significantly greater HPWL than R-SAD.

Lemma 1. *If each MAC $v_{i,j}$ in an $m \times h$ sub-array is placed at y-coordinate $y_{(i,j)}$ in the DSP column following the order consistency constraint, the total HPWL is given by*

$$\sum_{j=1}^{h}(y_{(m,j)} - y_{(1,j)}) + \sum_{i=1}^{m}(y_{(i,h)} - y_{(i,1)}) \tag{2}$$

Proof. Omitted due to space limit. □

Lemma 1 shows that with the OC (Order Consistency) constraint, only the placement of the MACs in the bottom/top row $(y_{(1,j)}, y_{(m,j)})$ and left/right columns $(y_{(i,1)}, y_{(i,n)})$ contributes to the total HPWL. Minimizing HPWL implies minimizing the y coordinates of the MACs on the top row and right column, and maximizing the y coordinates of the MACs on the bottom row and left column. Please note that the MACs at the four corners don't affect HPWL since $y_{(1,1)} = 1$ and $y_{(m,h)} = m{\cdot}h$ due to Constraints 1 and 2, and $y_{(m,1)}$ and $y_{(1,h)}$ are canceled out in Equation (2).

R-SAD divides the given sub-array into 7 regions, and each region is placed independently. The region division and the placement order of the 7 regions are shown in Figure 7(a). The dimensions of the regions depend on an integer parameter $g \leq \min(m, h)/2$. The value of g is decided through a linear search according to corresponding placement HPWL, which can be calculated using Equation (1) without performing the actual placement. The row-by-row sweep method in Figure 7(b) is a special case of R-SAD for g=1.

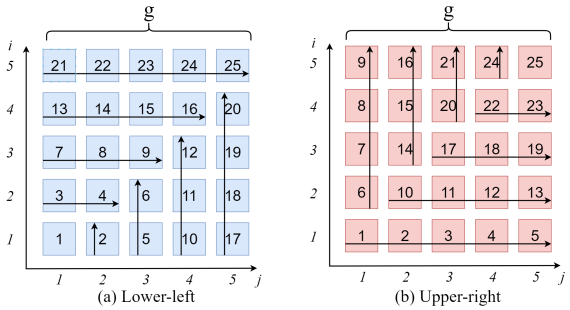


Figure 8: R-SAD placement of the MACs in the lower-left and upper-right regions. The numbers indicate *relative* placement order in y-direction.

Lower-left region: The R-SAD placement of the lower-left region, a $g \times g$ square, is illustrated in Figure 8(a), where the numbers indicate the order along y-direction in the DSP column after placement, e.g., the MAC with 9 is right above the MAC with 8 in the DSP column. Visually, the order follows sweeps alternating between horizontal and vertical directions. This order has three properties.

1) The R-SAD order conforms to the order consistency constraint.
2) For any sub-square starting from the lower-left corner, the upper-right corner MAC always has the maximum order index. For the lower-left 2×2 sub-square in Figure 8(a), its upper-right corner MAC has index 4, which is the maximum among the 4 MACs in this sub-square. Such index is the minimum that permits an order within the sub-square satisfying the order consistency constraint.
3) This order allows an analytical form expression for assigning the order index to each MAC.

For a MAC in the i-th row and j-th column of the lower-left region, its order index is given by

$$O_{ll}(i,j) = \begin{cases} i^2 - i + j & \text{if } i \geq j \\ (j-1)^2 + i & \text{if } i < j \end{cases} \tag{3}$$

For the diagonal MACs $v_{i,i}$, Equation (3) is reduced to $O_{ll}(i,i) = i^2$, which is the size of its lower-left sub-square. The diagonal MACs serve as anchors for deriving the order indices of the other MACs. For a MAC $v_{i,j}$ above the diagonal ($i > j$), MAC $v_{i,i}$ serves as the anchor and the index difference from the anchor equals the horizontal distance from the anchor. Therefore, $O_{ll}(i,j) = O_{ll}(i,i) - (i-j) = i^2 - i + j$. For MACs below the diagonal ($i < j$), $v_{j-1,j-1}$ is used as the anchor. The index difference between $v_{i,j}$ and $v_{j-1,j-1}$ is i and $O_{ll}(i,j) = O_{ll}(j-1, j-1) + i = (j-1)^2 + i$.

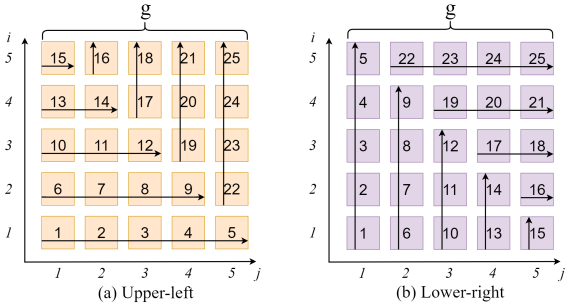


Figure 9: R-SAD placement of the MACs in the lower-right and upper-left regions. The numbers indicate *relative* placement order in y-direction.

Lower-right region: The R-SAD placement for the lower-right region is illustrated in Figure 9(b) and the order index is given by

$$O_{lr}(i,j) = \begin{cases} -\frac{1}{2}j^2 + (g+\frac{3}{2})j - g + i - 1, & i \le g - j + 1 \\ \frac{1}{2}g(g-1) + \frac{1}{2}i(i-1) + j, & i > g - j + 1 \end{cases} \quad (4)$$

Please note the row/column indices i/j and order index O_{lr} here are with respect to the region instead of the sub-array. For the example in Figure 7(a), the *absolute* placement order of the lower-right (purple) region ranges from 16 to 24, and the MAC placed at $y = 16$ has *relative* row/column indices $(1, 1)$.

Upper-left region: The order index is given by

$$O_{ul}(i,j) = \begin{cases} -\frac{1}{2}i^2 + (g+\frac{3}{2})i - g + j - 1, & i \le g - j + 1 \\ \frac{1}{2}g(g-1) + \frac{1}{2}j(j-1) + i, & i > g - j + 1 \end{cases} \quad (5)$$

Upper-right region: The order index is given by

$$O_{ur}(i,j) = \begin{cases} 2g \cdot j - j^2 + i - g & \text{if } i \ge j \\ 2g \cdot i - i^2 + i + j - 2g & \text{if } i < j \end{cases} \quad (6)$$

Lower-middle and upper-middle regions: The MACs in the lower-middle and upper-middle regions are placement in column-by-column sweeps. The order index is given by

$$O_{lm}(i,j) = O_{um}(i,j) = (j-1)g + i \quad (7)$$

Central region: The MACs in the $(m-2g)\times g$ central region are placed in row-by-row sweeps with the order index given by

$$O_{ctr}(i,j) = (i-1)h + j \quad (8)$$

From Figure 8 and 9 one can see that the R-SAD placement in the four corner regions are actually in the same pattern and is symmetric to the center of the MAC sub-array.

Lemma 2. *The intra-column HPWL obtained from R-SAD placement of a $m \times h$ sub-array is given by Equation* (1)*.*

Proof. Equation (1) is derived by integrating Equations (3) - (8) and the details are omitted due to space limit. □

Lemma 3. *The computational complexity of R-SAD is $\mathcal{O}(1)$ for a 2×2 or larger sub-array ($m, h \ge 2$).*

Proof. The RHS of Equation 1 is a third-degree polynomial function of g. The critical points of it can be calculated by setting the derivative to zero:

$$g_{1,2} = h \pm \sqrt{\frac{1}{2}(h - \frac{1}{2})^2 + \frac{5}{24}}, \quad (9)$$

where g_1 reaches local minimum and g_2 reaches local maximum. It can also be proved that $0 < g_1 < \frac{h}{2} < g_2$ for $h \ge 2$. Since g is an integer in $[1, \frac{\min(m,h)}{2}]$, the local minimum becomes the minimum in $[1, \frac{\min(m,h)}{2}]$. Therefore, the best g with minimum HPWL can be found in $\{\lceil g_1 \rceil, \lfloor g_1 \rfloor, \lfloor \frac{m}{2} \rfloor\}$. □

3) Partition Candidate Placement Using R-SAD: A partition candidate consists of s sub-arrays, each of which is placed on a single DSP column using R-SAD. According to Algorithm 1, the first $s-1$ sub-arrays always have the same width (number of MAC columns) while the last one may the same or smaller width. For sub-arrays of the same width, we index them from left to right according to the given MAC array order. R-SAD is performed for the first sub-array, and the solution is duplicated for all sub-arrays with odd-numbered indices. Meanwhile, the solution of the first sub-array is mirrored and then applied to all sub-arrays with even-numbered indices. By doing so, the solutions of every two neighboring sub-arrays are mirrored from each other. In Figure 10, the placement of the second sub-array is mirrored from the placement of the first sub-array. One can see that two MACs with inter-column connection always have the same y-coordinate (horizontally aligned) in their DSP columns. As such, there is no vertical wire for inter-column connections. This property not only minimizes inter-column wirelength, but also facilitates the HPWL bound estimation in Step 2 (Section VI-C). If the right-most sub-array has a smaller width, dummy MAC columns are padded to it in the same way in HPWL upper bound calculation.

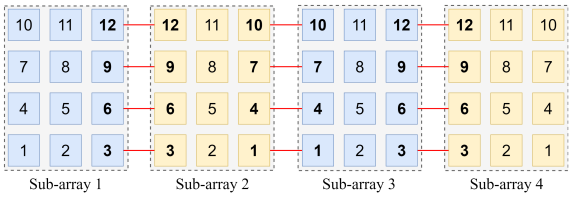


Figure 10: R-SAD placement of an 8 x 12 MAC array partitioned to four 8 x 3 sub-arrays. The numbers indicate y-coordinates in DSP columns. The red lines indicate inter-column wires.

VII. Experiments

A. Testcases and Experiment Setup

Table II: Characteristics of testcases.

Designs	PE Array Size	#Cells	#DSPs	Clock Period (ns)
IS8	8 x 8	4.4K	64	4
OS8	8 x 8	5.5K	64	4
IS16	16 x 16	16K	256	5
OS16	16 x 16	20K	256	5
IS32	32 x 20	35K	640	6
OS32	32 x 20	40K	640	6

Six systolic array-based CNN circuits are designed according to [3]. The characteristics of these testcases are summarized in Table II. These cases cover both *Input Stationary (IS)* and *Output Stationary (OS)* designs. They are placed onto two different FPGA architectures, *FPGA1* and *FPGA2*. FPGA1 is adopted from the ISPD 2016 placement contest [13] and has 678 DSPs distributed in four columns. FPGA2 is a larger architecture with 1800 DSPs distributed in five columns. The clock period constraints are generated in a way such that the resulting worst slacks are slightly less than zero when using Vivado placement in timing driven mode. The experiments are performed on a computer with a 3.80 GHz CPU and 32 GB RAM.

In the experiment, we compare the following methods.

- Our **R-SAD** for MAC array placement. It is combined with other placers as R-SAD+Vivado, R-SAD+elfPlace and R-SAD+QP to generate the placement of entire CNN circuits.
- **Vivado**, an industrial FPGA tool [14]. It is used to place entire CNN circuits or the non-MAC parts of circuits. We run Vivado Design Suite 2018.3 in both timing-driven mode and non-timing-driven mode, while the other parameters being in default values.
- **QP**, a quadratic placer based on UTPlaceF [15]. It is used to place entire CNN circuits, only place MAC arrays in QP+Vivado, or only place the non-MAC parts in R-SAD+QP.
- **elfPlace [10]**, a state-of-the-art placer. It is based on the same algorithm as DreamPlaceFPGA [11] but runs on CPU. It is used to place entire CNN circuits, only place MAC arrays in elfPlace+Vivado, or place the non-MAC parts in R-SAD+elfPlace.
- **EA**, the evolutionary algorithm using NSGA-II [16] package as in RapidLayout [7] for MAC array placement. We use the same genotype and parameter settings as in [7]. MAC array HPWL

Table I: HPWL ($\times 10^5$) and placement runtime (seconds) with and without R-SAD-based MAC array placement.

Device	Design	Vivado		R-SAD + Vivado		elfPlace		R-SAD + elfPlace		QP		R-SAD + QP	
		HPWL	CPU	HPWL	CPU	HPWL	CPU	HPWL	CPU	HPWL	CPU	HPWL	CPU
FPGA1	IS8	0.25	47	0.19	45	0.37	146	0.20	141	0.25	9	0.21	8
	OS8	0.25	61	0.21	44	0.29	129	0.28	115	0.30	11	0.25	11
	IS16	1.24	101	0.92	104	0.82	124	0.75	116	1.36	28	0.88	25
	OS16	1.19	110	0.90	102	0.91	171	0.81	113	1.71	39	0.97	38
	IS32	3.91	207	2.36	219	2.73	194	1.94	184	4.94	62	2.15	58
	OS32	3.43	199	2.76	190	2.73	128	2.14	123	4.61	99	3.48	89
	Norm.	1.00	1.00	0.71	0.97	1.00	1.00	0.78	0.89	1.00	1.00	0.60	0.92
FPGA2	IS8	0.26	59	0.23	76	0.21	142	0.20	237	0.25	9	0.21	12
	OS8	0.25	61	0.25	75	0.21	135	0.19	204	0.28	12	0.24	15
	IS16	1.08	120	0.90	102	2.71	255	0.69	182	1.21	42	0.85	37
	OS16	1.18	116	0.89	111	1.72	229	0.78	237	1.49	36	0.93	52
	IS32	2.22	216	1.66	202	3.26	297	1.89	286	3.51	57	2.43	60
	OS32	2.75	222	2.26	200	2.77	305	1.89	256	4.39	76	1.96	72
	Norm.	1.00	1.00	0.80	0.96	1.00	1.00	0.52	1.03	1.00	1.00	0.60	1.07

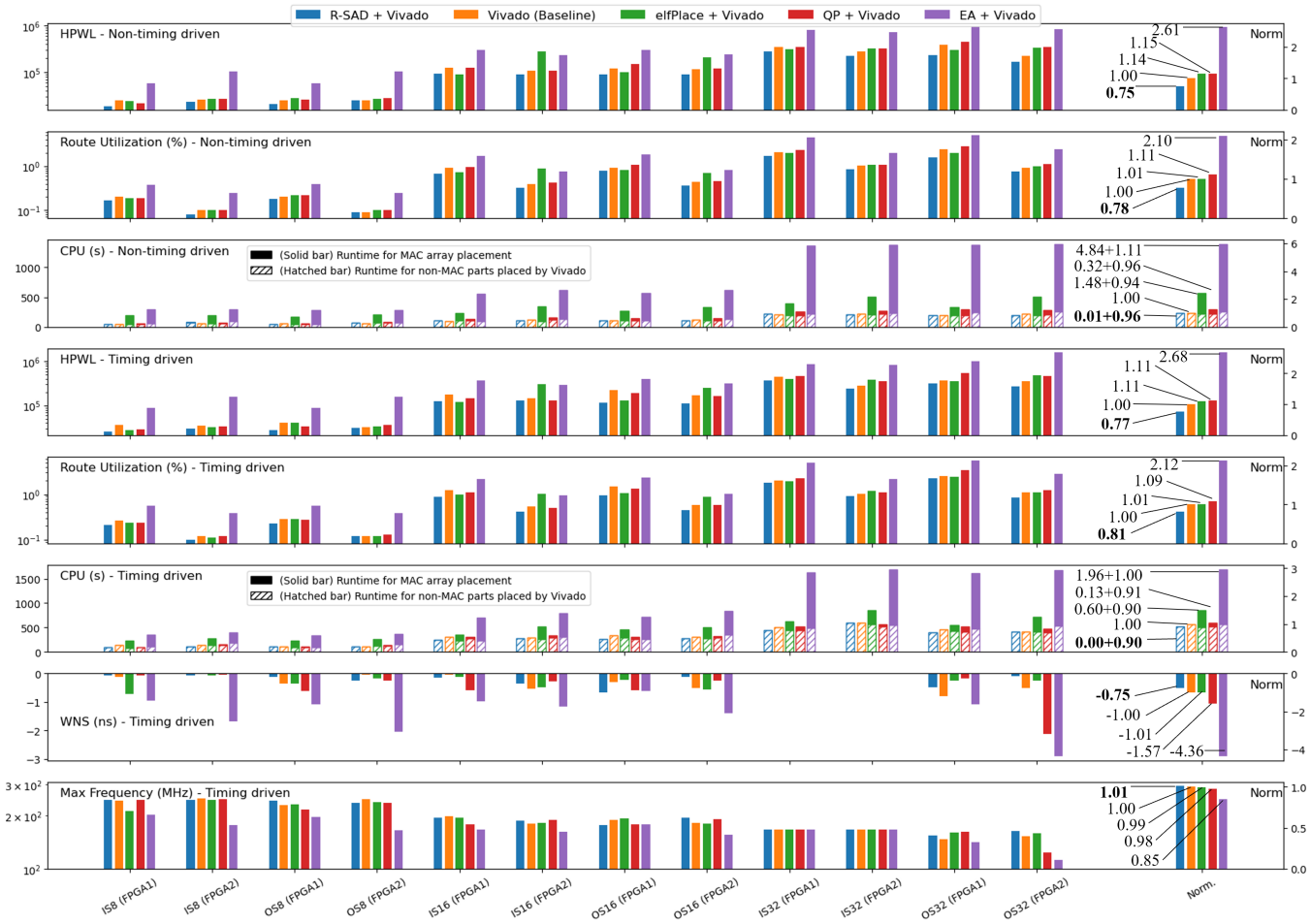


Figure 11: Vivado place & route results with different MAC array placements. Vivado runs in timing-driven and non-timing-driven modes.

serves as the objective function. It is performed with Vivado in EA+Vivado to generate placement for entire circuits as in [7].

When QP and elfPlace are employed for only MAC array placement, we run their global placement for the entire circuit. Then, the MAC locations in the global placement results are retained, and the rest circuits are placed by Vivado. The description of the greedy floorplan constraint approach in [4] lacks sufficient details for replicating its implementation.

B. Main Results

Table I lists the HPWL and placement runtime of Vivado (non-timing-driven), elfPlace, and QP with and without our R-SAD-based MAC array placement. Metrics are normalized separately for each placer to demonstrate the improvement achieved by R-SAD. It shows that R-SAD reduces the HPWL of Vivado placement by 20% and 29% on two FPGAs, and also decreases total runtime. For elfPlace and QP, applying R-SAD reduces HPWL by 22%-48%, with similar placement runtime. The runtime of R-SAD alone is always less than 70 milliseconds for every case, which is negligible compared to the total placement runtime.

In Figure 11, we compare our R-SAD+Vivado approach with other methods, most of which have MAC array placed by EA, elfPlace and QP. The Vivado placement and routing are performed in both timing-

driven and non-timing-driven modes. The results are evaluated by HPWL, routing resource utilization, placement runtime (MAC array + non-MAC parts), worst negative slack, and maximum frequency of the routed designs. One can see that the solutions from EA, which is similar to RapidLayout [7], are generally not competitive. Our R-SAD+Vivado approach always achieves the best average results on all these metrics. Besides the **23% - 25% reduction on HPWL** compared with Vivado, our approach usually reduces routing resource utilization by 19% or more. This implies that our approach helps routability significantly. Combined with R-SAD, the placement runtime of Vivado reduces 3% - 10%. In the timing-driven mode, our approach obtains the least worst negative slack and the maximum frequency on average.

C. Comparison with ILP and QAP Solver

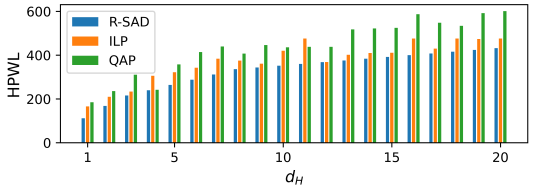


Figure 12: HPWL of an 8x8 MAC array placed by R-SAD, ILP, and QAP solvers with different d_H (distance between DPS columns).

The MAC array placement problem can be formulated as an ILP problem or a QAP (Quadratic Assignment Problem) [17]. We use ILP solver Gurobi [18] and QAP solver with the FAQ method [19] from SciPy [20] library to place an 8×8 MAC array with different distances between two neighboring FPGA columns. The results are shown in Figure 12. For all these cases, the ILP solver spends 1 hour runtime without finding the optimal solution, while the QAP solver finishes in 30–60 seconds without guaranteeing optimality either. One can see that R-SAD always achieves shorter HPWL than both ILP and QAP. Moreover, R-SAD is $50K\times$ faster than the ILP solver.

D. Effect of the Order Consistency Constraint

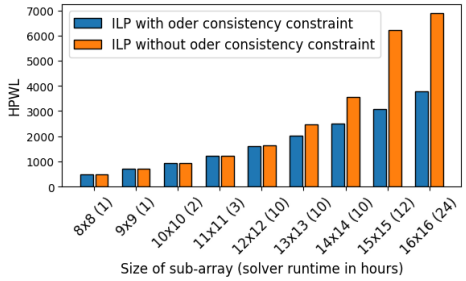


Figure 13: HPWL of MAC sub-arrays placed by ILP solver with and without the order consistency constraint.

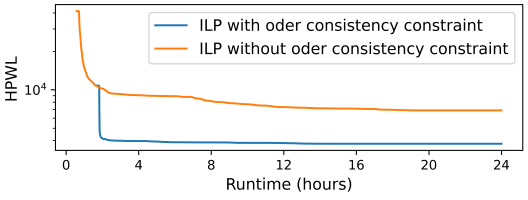


Figure 14: HPWL of 16×16 MAC sub-array placed by ILP solver with and without the order consistency constraint.

To examine the effect of the OC (Order Consistency) constraint proposed in Section VI-D1, we use the ILP solver to place MAC sub-arrays with and without the OC constraint. The runtime of the ILP solver is set to be 24 hours. Figure 13 shows the HPWL of MAC sub-arrays of different sizes after placement. One can see that the OC constraints facilitate significantly optimizing HPWL for large cases and do not affect the results for small cases. Figure 14 shows the HPWL changes over the runtime for a 16×16 sub-array. With the OC constraints, the solver converges much faster, while the solution is much worse without the constraints.

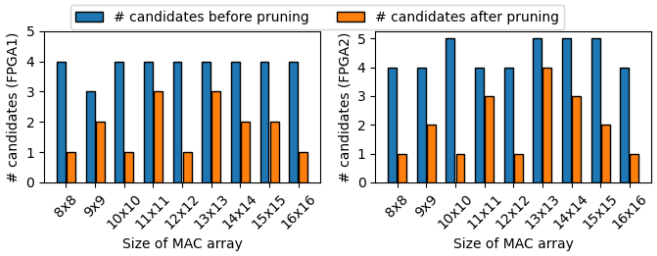


Figure 15: Number of candidates before and after pruning (Step 2).

E. Effect of Partition Candidate Pruning

To show the effectiveness of Step 2 (partition candidate pruning), we list the number of partition candidates before and after pruning for the two FPGA architectures in Figure 15. On average, 55% of partition candidates are pruned out in Step 2, hence about a half of the downstream computation workload is reduced. Please note that Step 1 (partition candidate generation) implicitly prunes out candidates that do not satisfy the properties mentioned in Section VI-B. Combining Steps 1 and 2, the percentage of pruned candidates is even higher.

F. Best Value for g in R-SAD

Figure 16 shows HPWL versus g for placing an $m\times h$ sub-array onto one DSP column by R-SAD. The curves correspond to the RHS of Equation (1). It can be seen that HPWL is convex in the search range $g \in [1, \frac{min(m,h)}{2}]$ (solid line). Therefore, the best g for the minimum HPWL can be determined in constant time.

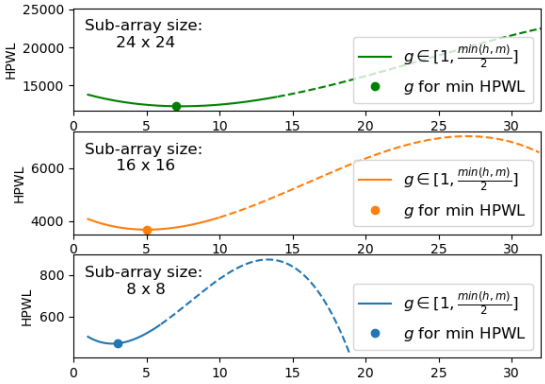


Figure 16: HPWL of R-SAD placement v.s. g for different sub-arrays.

VIII. Conclusions

In this paper, we have presented an algorithmic approach for systolic array placement on FPGAs. In particular, we propose a Region-wise Sweep in Alternating Direction (R-SAD) algorithm to place MAC sub-arrays on a single DSP column with empirically minimum intra-column wirelength. A partition enumeration and pruning framework is developed to utilize R-SAD for 2D MAC array placement. With our framework being integrated with general FPGA placement tools, we demonstrate 23%-25% reduced HPWL as well as improvement in routing resource utilization, accelerated placement runtime, and timing performance.

REFERENCES

[1] H. T. Kung and C. E. Leiserson, "Systolic arrays (for VLSI)," *Sparse Matrix Proceedings*, pp. 256–282, 1979.

[2] N. P. Jouppi, C. Young, N. Patil, D. A. Patterson, G. Agrawal, R. S. Bajwa, S. Bates, S. Bhatia, N. J. Boden, A. Borchers, R. Boyle, P. luc Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. B. 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. A. 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.

[3] 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," *Design Automation Conference*, pp. 1–6, 2017.

[4] J. Zhang, W. Zhang, G. Luo, X. Wei, Y. Liang, and J. Cong, "Frequency improvement of systolic array-based cnns on FPGAs," *International Symposium on Circuits and Systems*, pp. 1–4, 2019.

[5] D. Fang, B. Zhang, H. Hu, W. Li, B. Yuan, and J. Hu, "Global placement exploiting soft 2D regularity," *International Symposium on Physical Design*, p. 203–210, 2022.

[6] H. Kong, L. Feng, C. Deng, B. Yuan, and J. Hu, "How much does regularity help FPGA placement?" *International Conference on Field-Programmable Technology*, pp. 76–84, 2020.

[7] N. Zhang, X. Chen, and N. Kapre, "RapidLayout: Fast hard block placement of FPGA-optimized systolic arrays using evolutionary algorithms," *International Conference on Field-Programmable Logic and Applications*, pp. 145–152, 2020.

[8] W. Li, Y. Lin, M. Li, S. Dhar, and D. Z. Pan, "UTPlaceF 2.0: A high-performance clock-aware FPGA placement engine," *Transactions on Design Automation of Electronic Systems*, pp. 1 – 23, 2018.

[9] G. Chen, C.-W. Pui, W.-K. Chow, K.-C. Lam, J. Kuang, E. F. Y. Young, and B. Yu, "RippleFPGA: Routability-driven simultaneous packing and placement for modern FPGAs," *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 2022–2035, 2018.

[10] Y. Meng, W. Li, Y. Lin, and D. Z. Pan, "elfPlace: Electrostatics-based placement for large-scale heterogeneous FPGAs," *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 155–168, 2022.

[11] R. S. Rajarathnam, M. B. Alawieh, Z. Jiang, M. A. Iyer, and D. Z. Pan, "DREAMPlaceFPGA: An open-source analytical placer for large scale heterogeneous FPGAs using deep-learning toolkit," *Asia and South Pacific Design Automation Conference*, pp. 300–306, 2022.

[12] Y. G. Saab, "An improved linear placement algorithm using node compaction," *Transactions on computer-aided design of integrated circuits and systems*, pp. 952–958, 1996.

[13] S. Yang, A. Gayasen, C. Mulpuri, S. Reddy, and R. Aggarwal, "Routability-driven FPGA placement contest," *International Symposium on Physical Design*, 2016.

[14] Tom Feist, "White paper: Vivado design suite," 2023. [Online]. Available: https://docs.xilinx.com/v/u/en-US/wp416-Vivado-Design-Suite

[15] W. Li, S. Dhar, and D. Z. Pan, "UTPlaceF: A routability-driven FPGA placer with physical and congestion aware packing," *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 869–882, 2018.

[16] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *Transactions on Evolutionary Computation*, pp. 182–197, 2002.

[17] T. C. Koopmans and M. J. Beckmann, "Assignment problems and the location of economic activities," *Econometrica*, p. 53, 1957.

[18] Gurobi Optimization, LLC, "Gurobi optimizer reference manual," 2023. [Online]. Available: https://www.gurobi.com

[19] J. T. Vogelstein, J. M. Conroy, V. Lyzinski, L. J. Podrazik, S. G. Kratzer, E. T. Harley, D. E. Fishkind, R. J. Vogelstein, and C. E. Priebe, "Fast approximate quadratic programming for graph matching," *PLOS ONE*, 2015.

[20] E. Jones, T. Oliphant, P. Peterson *et al.*, "SciPy: Open source scientific tools for Python," 2001–. [Online]. Available: http://www.scipy.org/