Factored Radix-8 Systolic Array for Tensor Processing*

Inayat Ullah
Incheon National University

Kashif Inayat
Incheon National University

Joon-Sung Yang
Yonsei University

Jaeyong Chung†
Incheon National University

ABSTRACT

Systolic arrays are re-gaining the attention as the heart to accelerate machine learning workloads. This paper shows that a large design space exists at the logic level despite the simple structure of systolic arrays and proposes a novel systolic array based on factoring and radix-8 multipliers. The factored systolic array (FSA) extracts out the booth encoding and the hard-multiple generation which is common across all processing elements, reducing the delay and the area of the whole systolic array. This factoring is done at the cost of an increased number of registers, however, the reduced pipeline register requirement in radix-8 offsets this effect. The proposed factored 16-bit multiplier achieves up to 15%, 13%, and 23% better delay, area, and power, respectively, compared with the radix-4 multipliers even if the register overhead is included. The proposed FSA architecture improves delay, area, and power up to 11%, 20% and 31%, respectively, for different bitwidths when compared with the conventional radix-4 systolic array.

CCS Concepts

•**Computer systems organization** → **Systolic arrays;**

Keywords

Machine Learning, Systolic Arrays, Booth Multipliers

1. INTRODUCTION

In recent years, deep learning has demonstrated the predictive performance unbeatable by any other known methods. Deep neural networks have replaced many hand-crafted algorithms in various fields including computer vision, image/video compression, natural language processing, reinforcement learning, etc., [1–4]. However, deep neural networks require a massive amount of computation, which hinders the wide deployment of the models at various devices and slows down the innovations in the field of artificial intelligence. Thus, the demand for more compute power is higher than ever before and custom hardware to accelerate deep learning inference and training is being studied actively [5–7].

In most machine learning models including deep learning, the matrix multiplication is the key primitive. Most computations required by the models are explicitly represented in matrix multiplications, or transformed into them easily [8,9]. For example, 2D convolutions in a convolutional layer can be transformed into a set of matrix multiplications using Winograd algorithm [8], and this also reduces the number of multiplications. Owing to this property, custom accelerators for machine learning can provide the matrix multiplication as the only data processing instruction at the software-hardware interface, and faster speed and high efficiency can be achieved by eliminating architectural features included to support general-purpose processing. In such type of computing systems, the datapath design for computer arithmetic deserves more attention and is even more important than that in the conventional microprocessors.

The systolic array (SA) is a parallel computer architecture and consists of processing elements (PEs) organized as a linear or two-dimensional array. It is configured or hard-wired for specific operations such as matrix multiplications. The systolic array was invented back in 1979 by H.T. Kung and Charles E. Leiserson [10,11] and re-gained the attention when it was employed in several accelerators to speed up deep learning [5,12]. The first version of TPU (TPUv1) [5] used a 256×256 8-bit integer SA, designed to accelerate the inference. A TPUv2 chip consists of two cores, each containing 128×128 SA, while each of the two cores in TPUv3 chip deploys two 128×128 SAs [13]. They are used to speed up the training as well as the inference. The authors at IBM [12] implemented a 28×28 wavefront SA on an FPGA to accelerate the training.

With increasing interest in systolic arrays, many studies have been performed on systolic arrays [5,6,14–17], but to the best of our knowledge, none of them deal with the logic-level design of systolic arrays. This may be because multipliers in systolic arrays have been considered an atomic black-box primitive. This paper deals with the logic-level design of systolic arrays considering the structure of the multipliers together. The major contributions of this paper can be summarized as follows.

- We present the concept of *factored systolic arrays* and show that a large design space exists at the logic level despite the simple structure of the systolic arrays.
- We propose a realization of the factored systolic arrays based on radix-8 multipliers and demonstrates its benefits in area, delay, and power.
- We also show that the radix-8 multiplier uses much fewer pipeline registers compared with the conventional

*This work was supported by Samsung Research Funding & Incubation Center of Samsung Electronics under Project Number SRFC-TB1803-02.
†Correspondence to: Jaeyong Chung<jychung@inu.ac.kr>.

978-1-7281-1085-1/20/$31.00

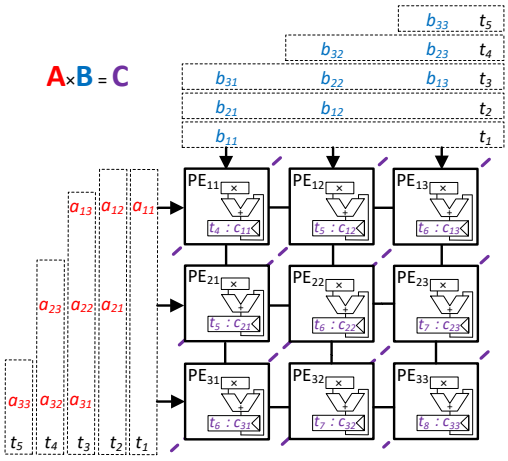

Figure 1: A 3×3 systolic array for matrix multiplication.

radix-4 multiplier, reducing or offsetting the cost of the proposed factoring.

2. BACKGROUND

2.1 Systolic Array Architecture

We consider a 2-dimensional (2-D) systolic system consisting of a set of PEs interconnected as a 2D-array. Figure 1 shows a simplified example of a 3×3 systolic array performing the multiplication of two 3×3 matrices A and B and producing resultant matrix C of the same size. The rows of the input matrix A are provided to the PEs on the left edge and passed to the right. The columns of the matrix B are provided to the PEs on the top and passed downward. Each PE contain a multiplication and accumulation (MAC) unit which performs the multiplication operation and the product is accumulated in the accumulator register to make the partial result. Every cycle the next batch of the inputs is provided to the edge PEs and the previous inputs are forwarded to the next PEs. In the example given the PE_{11} performs multiplication operation on the inputs a_{11} and b_{11} in clock cycle t_1. In the second clock cycle t_2 PE_{11} performs the multiplication on the next inputs a_{12} and b_{21} and forwards the previous cycle inputs a_{11} and b_{11} to PE_{12} and PE_{21}, respectively. In the same clock cycle PE_{12} and PE_{21} performs multiplication operations on the pairs (a_{11}, b_{12}) and (a_{21}, b_{11}), respectively. In the third clock cycle t_3, PE_{11} performs the multiplication operation on the third inputs a_{13}, b_{31} and outputs the final result c_{11} in fourth cycle t_4. The PE_{12} and PE_{21} outputs the resultant elements of matrix C c_{12} and c_{21} in fifth clock cycle. In this way, the values of the input matrices are propagated in systolic array and the output matrix C elements are computed in a wave-front flow as shown in the Figure 1. In the same manner this 3×3 systolic array is able to perform the multiplication of two $3 \times k$ and $k \times 3$ size matrices.

2.2 Related Works

Our work is mainly related to systolic array architecture and MAC unit design. Systolic arrays are usually designed for specific operations such as convolution [6,14–16] and matrix multiplications [5, 17]. Due to the popularity of convolutional neural networks (CNNs), majority of works target convolutions. Most of the existing studies on SA are performed at **architectural level**. They aim at minimizing memory access to reduce energy consumption and design dataflows to maximize data reuse and local accumulation. In [6], the authors target at accelerating the convolution operations in CNNs, study existing dataflows such as no local reuse, weight stationary, output stationary, and proposes a novel dataflow called row stationary. In [14], data reuse in the row stationary dataflow is further exploited by sharing the storage of processing elements. In [17], dataflows of systolic arrays for matrix multiplication are studied. Several studies [15, 16] also consider the physical mapping of processing elements into 3D ICs and enable data reuse along the third dimension, achieving further speed-up and less energy consumption. Unlike these studies, our study deals with **logic level** design of systolic arrays. To the best of our knowledge, none of the existing works on SAs are studied at this level. We target the systolic array for matrix multiplication which provides the same functionality as the conventional systolic array and achieves significant improvements in area, power, and delay.

Since the MAC unit is a fundamental building block of deep learning accelerators, studies on the unit [18, 19] have become active recently. In [18], using the weight sharing property of CNNs, the authors propose parallel accumulate shared MAC, which counts the frequency of each weight and the final accumulated value is calculated in a subsequent multiply phase. Unlike this design, our proposed design does not impose any functional limitations. In [19], the authors show that some pipeline registers in the MAC unit can be eliminated because only the final value is used out of the large number of multiplyaccumulations in machine learning applications. Our proposed design is orthogonal to this approach. Unlike these MAC studies, we deal with the entire systolic array.

3. PROPOSED DESIGN

3.1 A General Idea for Systolic Arrays

We consider a systolic array where PEs are organized in a grid. The systolic array accepts the inputs for a matrix at the left edge and the inputs for the other matrix at the top edge, and produces the outputs at the top edge. Each PE consists of a MAC which takes two inputs X and Y and performs the multiplication and accumulation operations every clock cycle. The input of the MAC fed from left side are considered as multiplier X, and the input fed from the top as multiplicand Y. Let P be the value stored in the accumulator when the accumulation is done and Q be the dot-product (i.e., the sum of the products). We assume that X, Y, and Q are represented in binary for compatibility but P can be represented in any format. The MAC unit performs the dedicated pre-processing on the X and Y prior to the main processing. The P-dedicated post-processing takes the MAC output and produces Q. The widely employed modified booth encoding multiplier in conventional general-purpose processors performs booth recoding as the X-dedicated pre-processing. In the systolic array, the dedicated logic (DL) circuits for X-, Y- and P- dedicated processing are replicated across PEs and we can factor them out and can be placed at the inputs and the outputs of the array. Then, the area of the XDL (YDL and PDL) circuits can be amortized by the PEs in a row (column) and becomes marginal as the size of the array grows. Also, the delay of the DL circuits can be eliminated from the critical path. We call this structure the *factored systolic array*. The logic to convert the number format or for rounding may be placed naturally at the inputs and the

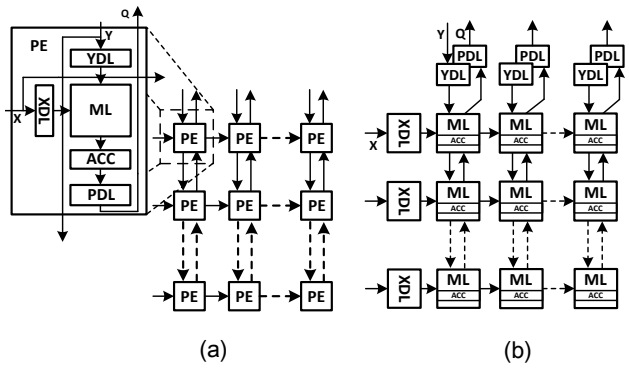

Figure 2: (a) Conventional systolic array architecture (b) Proposed factored systolic array architecture. XDL, YDL, and PDL are replicated across PEs and can be extracted and placed the inputs and the outputs of the systolic array, which allow us to re-think about what the best MAC design is for systolic arrays.

outputs, but in the factored systolic array, we extract logic circuits that are traditionally considered a part of the MAC unit such as booth encoders. In this structure, more aggressive, non-conventional approaches can be employed.

3.2 Parallel Multipliers in SAs

Multipliers dominate the datapath of the systolic array architecture. A parallel multiplier consists of three major parts: 1) partial product generation; 2) partial products reduction trees (e.g., Wallace tree); and 3) addition of the pair of final sum and carry rows of partial products using carry propagate adder (CPA) to get the final product.

Consider an $M \times N$-bit multiplier such that $X = x_{M-1}x_{M-2} \cdots x_1x_0$ and $Y = y_{N-1}y_{N-2} \cdots y_1y_0$ are the multiplier and multiplicand, respectively. The N rows of M-bit partial products $\mathrm{PP}_{i,j}$ can be expressed as

$$PP_{i,j} \;=\; x_i y_j, \;\; \forall i,j, \; 0 \le i < M, \; 0 \le i < N, \quad (1)$$

which is taken from [20]. In fast multipliers, the modified Booth encoding algorithm is employed to reduce the height of partial products [21, 22]. A radix-R=2^r, r>0, booth multiplier reduces the height of partial products from N to $\lceil (N+1)/r \rceil$, reducing the size and enhancing the speed of the reduction tree. The existing systolic array architectures employ radix-4 booth multipliers which has $\lceil (N+1)/2 \rceil$ number of partial products.

3.3 Radix-8 Booth Multiplier

In radix-8 booth multipliers the height of the partial products reduces to $\lceil (N+1)/3 \rceil$, which brings a noticeable reduction in the area of partial products reduction tree and critical path delay. However, booth multipliers with a radix higher than 2 requires hard multiples, that is not a power of two and are not obtainable using simple shift and complement operations. In radix-8 booth multiplier $\pm 0, \pm Y, \pm 2Y, \pm 3Y, \pm 4Y$ multiples of the multiplicand Y are required, and the generation of hard multiple $3Y$ involves the addition of Y and $2Y$ using CPA which slows down the multiplier.

The radix-8 booth recoding partitions the multiplier X into sets of 4 bits, 3 adjacent binary bits $x_{3(i+1)-1}x_{3i+1}x_{3i}$ and a borrow bit x_{3i-1} from the previous set, each set encoded into the control lines s, d, t, q, n using

$$s_i \;=\; \overline{(x_{3(i+1)-1} \oplus x_{3i+1})} \wedge (x_{3i} \oplus x_{3i-1})$$

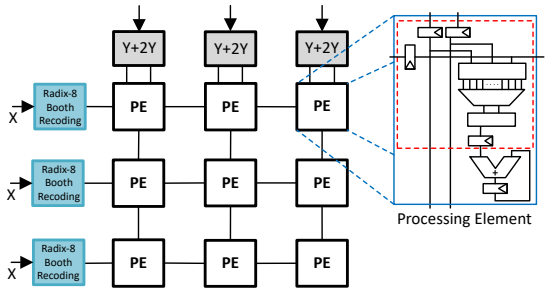

Figure 3: Factored radix-8 systolic array architecture. The circuit in the red-dotted box is considered the multiplier, which includes the input and the output registers.

$$\begin{aligned} d_i &= (x_{3i} \oplus x_{3i+1}) \wedge \overline{(x_{3i} \oplus x_{3i-1})} \\ t_i &= (x_{3(i+1)-1} \oplus x_{3i+1}) \wedge (x_{3i} \oplus x_{3i-1}) \\ q_i &= (x_{3(i+1)-1} \oplus x_{3i+1}) \wedge \overline{(x_{3(i)} \oplus x_{3i+1})} \wedge \overline{(x_{3i} \oplus x_{3i-1})} \\ n_i &= x_{3(i+1)-1} \\ \text{for } i &= 0,\ 1,\ 2, \cdots, \left\lceil \frac{N}{3} \right\rceil - 1, \end{aligned} \quad (2)$$

where N is the width of multiplier Y and $x_{-1} = 0$. These five control lines select a multiple of the multiplicand Y among $\pm 0, \pm Y, \pm 2Y, \pm 3Y, \pm 4Y$. For an $M \times N$ radix-8 multiplier, $5(\lceil \frac{N+1}{3} \rceil)$ control signals are generated in parallel. Except the all-zero case, these five control lines use one-hot encoding and we can re-design the code to reduce 5 lines to 4 lines. However, this can incur an additional delay in the booth selection and we use this standard coding as it is. The complexity of radix-8 booth recoding logic is higher than that of radix-4, but we will show that the entire booth recording logic can be removed from individual multipliers in a systolic array.

3.4 Factored Radix-8 Systolic Array (FSA)

A radix-8 booth multiplier involves the dedicated preprocessing of complex booth recoding on the multiplier X and $3Y = Y + 2Y$ generation on the multiplicand Y. When the radix-8 booth multiplier is employed in the PEs of the systolic array, the logic for the aforementioned dedicated preprocessing on X and Y are replicated across PEs. The main idea of this paper is to use the radix-8 booth encoding in designing the MACs of the systolic array and at the same time factor out this repeated dedicated pre-processing logic from the PEs and place them at the inputs.

Figure 3 shows the architecture of the proposed radix-8 FSA. The X inputs accepted from the left edge are preprocessed in parallel and the control lines for the booth selection are forwarded to the corresponding rows of PEs. The Y inputs accepted from the top edge are pre-processed and the resultant $3Y$s along with Ys are forwarded to the corresponding columns of PEs. In this way, the proposed FSA achieves the advantages of the -8 booth multipliers and at the same time eliminates the associated drawbacks by factoring out the complex booth encoding logic and hard multiple $3Y$ computation. This brings a significantly decreasing impact on the delay, area and power of the whole systolic array.

However, this gain in the performance of the FSA is achieved at the cost of an increased number of input registers in the PEs. A PE in the radix-8 FSA registers and forwards $3Y$ along with multiplicand Y to the next PE compared with Y alone in the PEs of radix-4 systolic array. Similarly, the PEs of the radix-8 FSA registers and forwards $5(\lceil \frac{N+1}{3} \rceil)$ number

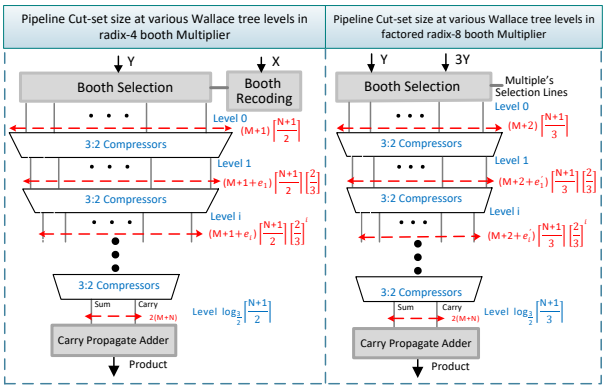

Figure 4: Pipeline cut-set cost comparison in proposed radix-8 and radix-4 multipliers. The pipeline cut-set size of the proposed multiplier is about 33% less than that of the conventional radix-4 multiplier at a corresponding level, reducing the number of required pipeline registers substantially.

Table 1: Number of partial product bits at each Wallace tree level in R4 and FR8 multipliers

Wallace tree level	8-bit		16-bit		32-bit	
	R4	FR8	R4	FR8	R4	FR8
Level-0	53	35	169	119	593	395
Level-1	43	31	131	94	431	300
Level-2	37	28	105	81	316	246
Level-3	29	-	92	69	259	191
Level-4	-	-	76	59	199	159
Level-5	-	-	62	-	168	123
Level-6	-	-	-	-	123	-

of control bits to the next PEs compared with N-bit multiplier X in the PEs of radix-4 systolic array. However, this increase in the number of input registers is compensated by the reduced pipeline cut-set registers cost in the multipliers of the proposed radix-8 FSA explained in the coming section.

3.5 Pipeline Cut-set Cost in Booth Multipliers

The computing systems for machine learning are usually highly optimized for throughput. Achieving a high throughput of the overall system necessitates the pipelining of the multipliers datapath. The partial products reduction tree makes the major part of a parallel multiplier. The pipelining of a multiplier's datapath requires the placement of balanced cut-set registers in the Wallace tree levels which is executed by registering all the partial product bits at that level. Thus, the number of partial product bits at any Wallace tree level decides the size of the pipeline cut-set. Figure 4 shows a summary of the pipeline cut-set sizes at various Wallace tree levels in radix-4 and factored radix-8 booth multipliers. Since shifted partial products of various widths are compressed at each Wallace tree level, the widths of the intermediate partial products vary. In Figure 4, e_i and e_i' denote positive integers and note that $e_i, e_i' \ll M$. Table 1 lists the actual number of partial product bits in 8, 16, and 32 bit multipliers. Figure 4 and Table 1 explains that the number of partial product bits in factored radix-8 multiplier are about 2/3 times lower than that of radix-4 and that the same ratio of 3 : 2 partial product bits is maintained at the same levels of Wallace trees in these multipliers. Thus,

Table 2: Performance comparison of radix-4, radix-8 and factored radix-8 multipliers.

WL	Design	Area (μm^2)	Power (μW)	Delay (ns)	PDP ($ns{\cdot}\mu W$)	ADP ($ns{\cdot}\mu m^2$)
8	R4	1869 (1.00)	0.95 (1.00)	1.12 (1.00)	1.06 (1.00)	2093 (1.00)
	R8	1931 (1.03)	0.85 (0.89)	1.05 (0.94)	0.89 (0.84)	2028 (0.97)
	FR8	1625 (0.87)	0.79 (0.83)	0.99 (0.88)	0.78 (0.74)	1609 (0.77)
16	R4	6413 (1.00)	3.44 (1.00)	1.48 (1.00)	5.09 (1.00)	9491 (1.00)
	R8	6416 (1.00)	3.36 (0.98)	1.49 (1.01)	5.01 (0.98)	9560 (1.01)
	FR8	5551 (0.87)	2.65 (0.77)	1.26 (0.85)	3.34 (0.66)	6994 (0.74)
32	R4	22926 (1.00)	12.10 (1.00)	1.29 (1.00)	15.61 (1.00)	29575 (1.00)
	R8	22196 (0.97)	11.30 (0.93)	1.36 (1.05)	15.37 (0.98)	30187 (1.02)
	FR8	18305 (0.80)	8.49 (0.70)	1.14 (0.88)	9.68 (0.62)	20868 (0.71)

the lower number of partial product bits in the Wallace tree of factored radix-8 significantly reduces its pipeline cut-set registers cost when compared with that of radix-4.

Moreover, the datapath of the factored radix-8 multiplier doesn't include booth encoding, hence, there is a strong tendency of placing a balanced pipeline cut-set at a lower level of the Wallace tree with further reduced number of partial products, thus, further reduction in the pipeline cut-sets cost. This reduced usage of pipeline registers in factored radix-8 multipliers offsets the aforementioned growth in its input registers in comparison with radix-4. It should be noted that the lower height of the factored radix-8 partial products along-with the extraction of booth encoding enables it in achieving lower critical path delay not only in the nonpipelined case but also ensures in the pipelined versions at a significantly low cost.

4. EVALUATION AND ANALYSIS

4.1 Evaluation Setup & Baselines

We use the conventional radix-4 (R4) and basic radix-8 (R8) multipliers and the corresponding systolic arrays as the baselines. We compare the proposed factored radix-8 (FR8) designs to the baselines at different wordlengths $WL = 8, 16$, and 32. A Wallace tree using CSAs (3 : 2 compressors) is used for the reduction of the partial products in designing multipliers. The multiplier designs include the input and the output registers. In a systolic array, the registers to propagate the (pre-preprocessed) inputs across PEs are the input registers of the multipliers, so *the factored multiplier design includes the cost as well as the benefit of the factoring.*

All the designs are described in Verilog and verified using Synopsys VCS. The designs are synthesized and mapped to an industrial 32nm standard cell library using the Synopsys Design Compiler. The post-synthesis gate-level simulation is performed again using Synopsys VCS and generates a switching activity interchange format (SAIF) file using random vectors. The power dissipation is estimated by the PrimeTime-PX tool by annotating the SAIF file to the netlist. The power consumption is measured at 1.5ns clock period unless stated otherwise. All the experiments are performed on a Linux machine with 128GB memory. This memory capacity limits the largest systolic array size in the experiments.

Table 3: Implementation results for 8, 16, and 32-bit systolic arrays of various sizes. The PDP and ADP of R8 are omitted due to limited space.

WL	Size	Radix-4 (R4)					Radix-8 (R8)			Proposed Design (FR8)				
		Area (mm^2)	Delay (ns)	Power (W)	PDP (ns·W)	ADP ($ns \cdot mm^2$)	Area (mm^2)	Delay (ns)	Power (W)	Area (mm^2)	Delay (ns)	Power (W)	PDP (ns·W)	ADP ($ns \cdot mm^2$)
8	16	0.54	1.15	0.26	0.30	0.62	0.55	1.14	0.24	0.50	0.99	0.23	0.23	0.50
	32	2.16	1.15	1.04	1.19	2.48	2.20	1.14	0.96	2.00	0.99	0.91	0.90	1.98
	64	8.63	1.15	4.15	4.77	9.93	8.78	1.14	3.83	7.96	0.99	3.63	3.59	7.88
	128	34.54	1.15	16.61	19.10	39.72	35.13	1.14	15.33	31.79	0.99	14.48	14.33	31.47
16	16	1.77	1.48	0.91	1.35	2.62	1.74	1.59	0.85	1.62	1.26	0.72	0.90	2.05
	32	7.09	1.48	3.64	5.38	10.50	6.96	1.59	3.39	6.47	1.26	2.85	3.59	8.16
	64	28.37	1.48	14.55	21.54	41.99	27.83	1.59	13.56	25.85	1.26	11.36	14.31	32.57
	128	113.48	1.48	58.15	86.06	167.95	111.33	1.59	54.19	103.28	1.26	45.38	57.18	130.14
32	16	5.89	1.37	3.14	4.31	8.06	5.67	1.36	2.88	4.72	1.22	2.17	2.65	5.76
	32	23.54	1.37	12.58	17.23	32.25	22.69	1.36	11.52	18.84	1.22	8.67	10.58	22.99
	64	94.16	1.37	50.29	68.90	129.00	90.78	1.36	46.09	75.26	1.22	34.63	42.25	91.82

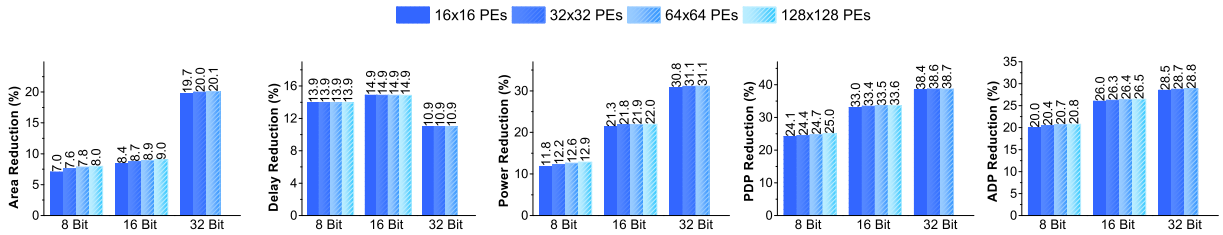

Figure 5: Performance improvements of the factored radix-8 systolic arrays with respect to the corresponding conventional radix-4 systolic arrays. The proposed design outperforms the conventional design in all metrics.

4.2 Multipliers

Balancing the area and the delay, we insert one pipeline cut-set to 8-bit and 16-bit multipliers and two pipeline cut-sets to 32-bit multipliers. The inserted pipeline registers are placed at optimal positions in terms of delay using retiming. Table 2 compares the area, power, delay, power delay product (PDP), and area delay product (ADP) of R4, R8, and FR8 multipliers. All the listed results in Table 2 are normalized with respect to those of the corresponding R4. Although R8 consumes little less power than R4, it does not provide significant benefits in area or power. This explains why R8 is not usually employed in practice. Compared to R8, FR8 does not have the adder for 3Y and the booth encoder. Compared to R4, FR8 has the reduced height of the partial products owing to radix 8 and has fewer pipeline registers. Therefore, although FR8 contains a larger number of the input registers than R4 and R8, it shows significant improvements in all the metrics over R4 and R8. The improvements are made across all the wordlengths. The area, power, and delay of FR8 are $13-20\%$, $17-30\%$, and $12-15\%$ less than R4, respectively. The PDP and ADP performance of FR8 is $26-38\%$ and $23-29\%$ better than those of R4.

4.2.1 *Sequential Area and Power Inversion*

The proposed factoring reduces delay and combinational area significantly, but it comes at the cost of an increased number of registers. Although the added registers take a small area compared to the reduced combinational area, registers take a significant portion of power consumption, and one can concern that the added registers may degrade overall power consumption. The idle power increase may be not significant owing to the clock-gating technique, but the active power can increase. However, our proposed FR8 design can have similar or smaller sequential area compared to R4

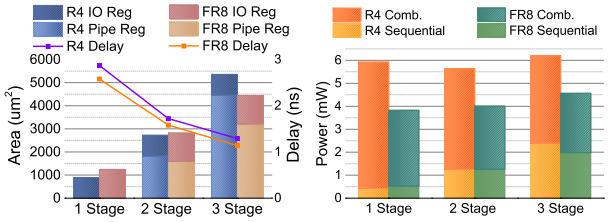

Figure 6: (a) Sequential area and delay. (b) Combinational and sequential power. When the number of pipeline stages increases, FR8 outperforms R4 even in terms of sequential area and sequential power.

because of fewer pipeline registers used. Figures 6a and 6b show the sequential area, delay, combinational power, and sequential power of 32-bit R4 and FR8 as the number of pipeline stages varies. The power consumption in this experiment are measured at $3ns$ clock period. The single stage design means the non-pipelined version. The delay is reduced substantially even when the number of pipeline stages increases from 2 to 3, which makes the 3-stage multiplier a reasonable design. Figure 6a also breaks the sequential area into the area of the input and output (IO) registers and the area of the pipeline registers. FR8 has a larger number of IO registers than R4 due to the factoring, resulting in a larger sequential area and a higher sequential power consumption in the non-pipelined version. However, when a pipeline cut-set is inserted for the two stage design, the overall sequential areas and powers become similar due to the fewer pipeline registers in FR4 than R8. In the three stage designs, the sequential areas and powers are finally inverted, making FR8 better in all aspects.

Table 4: Area and power breakdown of 16-bit 32×32 systolic arrays.

	Area by Module (mm^2)			Power by Component (W)		
R4	PE	7.09		ClkNet	1.1	(29.49%)
	⌞ Mult	6.11	(86.13%)	Reg	0.3	(9.43 %)
	⌞ Other	0.98	(13.87%)	Comb	2.2	(61.08%)
	Total	7.09		Total	3.6	
FR8	PE	6.45	(99.61%)	ClkNet	1.2	(42.38%)
	⌞ Mult	5.03	(77.68%)	Reg	0.3	(10.75%)
	⌞ Other	1.42	(21.93%)	Comb	1.3	(46.87%)
	BEncoder	0.01	(0.14%)			
	3YAdder	0.02	(0.25%)			
	Total	6.48		Total	2.8	

4.3 Systolic Arrays

Table 4 shows the area and the power breakdown for 16-bit R4 and FR8 systolic arrays of size 32×32. The R4 SA is just a grid of PEs and the PE area takes up the whole SA area. In R4 SA, the multipliers (Mult) occupies 86.13% of the whole SA area, so improvements in the multiplier design have significant impact on the large block of 7.09 mm^2. The area of the others (Other) including accumulators takes up the rest. The arithmetic circuits usually have a higher switching activity than control logic circuits, and the combinational (Comb) power consumption accounts for 61.08% of the total SA power consumption. The clock network (ClkNet) and registers (Reg) consumes 38.92%. In FR8, the booth encoder (BEncoder) and the 3Y adder (3YAdder) are extracted, but they together occupies just 0.39% area already in the 32×32 array. Compared to the R4, the sequential power (ClkNet+Reg) increases slightly by 0.1W, but the combinational power is reduced by 0.9W.

Table 3 compares the area, delay, power, PDP, and ADP of the R4, R8 and FR8 SAs at different sizes 16×16, 32×32, 64×64, and 128×128 and at different bitwidths WL=8, 16 and 32, while Figure 5 shows the improvement in the performance of the proposed FR8 SA over the conventional R4 SA. The delay of SAs doesn't depend on the SA size, so the delay reduction remains constant as the SA size changes. The area and the power reduction from R4 increases as the size of SA increases because the cost of the extracted logic is amortized over the PEs. The area reduction is prominent at 32-bit because the 3-stage pipeline is employed and FR8 reduces even the sequential area from R4. The delay reduction at 32-bit is less than that at 8-bit and 16-bit because the critical path appears in the accumulator. FR8 outperforms R4 in all metrics. The area, power, and delay of FR8 are $7.0 - 20.1\%$, $10.9 - 14.9\%$, $11.8 - 31.1\%$ less than those of R4, respectively. The PDP and ADP performance of FR8 is $24.1 - 38.7\%$ and $20.0 - 28.8\%$ better than those of R4, respectively.

5. CONCLUSIONS

We have presented a novel systolic array based on factoring and radix-8 and have demonstrated that the proposed design can achieve significant reductions in area, delay, and power from the conventional radix-4 design providing exactly the same functionality. The concept of the factored systolic array can be realized in many different ways and non-conventional multipliers such as radix-8 can be practical with the factored systolic array. We believe that more exploration is required in that research path, which may be our future work.

6. REFERENCES

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Adv Neural Inf Process Syst., 1097–1105,*, 2012.

[2] Oren Rippel and Lubomir Bourdev. Real-Time Adaptive Image Compression. In *Int Conf on Machine Learning-Vol. 70*, pages 2922–2930. JMLR. org, 2017.

[3] Tom Young, Devamanyu Hazarika, Soujanya Poria, and Erik Cambria. Recent trends in deep learning based natural language processing. *IEEE Comput. Intell Mag.*, 13(3):55–75, 2018.

[4] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

[5] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-Datacenter Performance Analysis of a Tensor Processing Unit. In *ACM/IEEE 44th Annual Int Symp on Comput Arch (ISCA)*, pages 1–12, 2017.

[6] Yu-Hsin Chen, Tushar Krishna, Joel S Emer, and Vivienne Sze. Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks. *IEEE Journal of Solid-State Circuits*, 52(1):127–138, 2016.

[7] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. EIE: Efficient Inference Engine on Compressed Deep Neural Network. In *Int Symp on Comput Archit (ISCA), pages=243–254, year=2016, organization=IEEE.*

[8] Andrew Lavin and Scott Gray. Fast Algorithms for Convolutional Neural Networks. In *Proc. of the IEEE Conf. on Comput Vis and Pattern Recognit*, pages 4013–4021, 2016.

[9] Minsik Cho and Daniel Brand. MEC: memory-efficient convolution for deep neural network. In *Proc. of the 34th Int Conf on Machine Learning-Volume 70*, pages 815–824. JMLR. org, 2017.

[10] HT Kung and CE Leiserson. Algorithms for VLSI processor arrays/eds. *C. Mead, L. Conway.–Addison-Wesley: Introduction to VLSI Systems*, 1979.

[11] Hsiang-Tsung Kung. Why systolic architectures? *IEEE computer*, 15(1):37–46, 1982.

[12] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. Deep Learning with Limited Numerical Precision. In *Int. Conf. on Machine Learning*, pages 1737–1746, 2015.

[13] Google. System Architecture. https://cloud.google.com/tpu/docs/system-architecture, 2019.

[14] Chen Xin, Qiang Chen, Miren Tian, Mohan Ji, Chenglong Zou, Xin'An Wang, and Bo Wang. Cosy: An Energy-Efficient Hardware Architecture for Deep Convolutional Neural Networks Based on Systolic Array. In *2017 IEEE 23rd Int Conf on Parallel and Distributed Systems (ICPADS)*, pages 180–189, 2017.

[15] Hsiang-Tsung Kung, Bradley McDanel, and Sai Qian Zhang. Mapping systolic arrays onto 3d circuit structures: Accelerating convolutional neural network inference. In *Int Workshop on Signal Processing Systems (SiPS)*, pages 330–336. IEEE, 2018.

[16] HT Kung, Bradley McDanel, Sai Qian Zhang, CT Wang, Jin Cai, CY Chen, Victor CY Chang, MF Chen, Jack YC Sun, and Douglas Yu. Systolic building block for logic-on-logic 3d-ic implementations of convolutional neural networks. In *2019 IEEE Int Symp on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2019.

[17] Ananda Samajdar, Yuhao Zhu, Paul Whatmough, Matthew Mattina, and Tushar Krishna. SCALE-Sim: Systolic CNN Accelerator Simulator. *arXiv preprint arXiv:1811.02883*, 2018.

[18] James Garland and David Gregg. Low Complexity Multiply Accumulate Unit for Weight-Sharing Convolutional Neural Networks. *IEEE Comput Archit L*, 16(2):132–135, 2017.

[19] Sungju Ryu, Naebeom Park, and Jae-Joon Kim. Feedforward-Cutset-Free Pipelined Multiply–Accumulate Unit for the Machine Learning Accelerator. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, 27(1):138–146, 2018.

[20] Alberto A Del Barrio, Román Hermida, and Seda Ogrenci-Memik. A Combined Arithmetic-High-Level Synthesis Solution to Deploy Partial Carry-Save Radix-8 Booth Multipliers in Datapaths. *IEEE Trans. Circuits Syst. I, Reg. Papers*, 66(2):742–755, 2018.

[21] Andrew D Booth. A signed binary multiplication technique. *The Quarterly Journal of Mechanics and Applied Mathematics*, 4(2):236–240, 1951.

[22] Milos D Ercegovac and Tomas Lang. *Digital Arithmetic*. Elsevier, 2004.