Double MAC: Doubling the Performance of Convolutional Neural Networks on Modern FPGAs

Dong Nguyen, Daewoo Kim and Jongeun Lee*
{dongnm, daewoo, jlee}@unist.ac.kr
School of Electrical and Computer Engineering, UNIST, Ulsan, Korea

***Abstract*—This paper presents a novel method to double the computation rate of convolutional neural network (CNN) accelerators by packing two multiply-and-accumulate (MAC) operations into one DSP block of off-the-shelf FPGAs (called *Double MAC*). While a general SIMD MAC using a single DSP block seems impossible, our solution is tailored for the kind of MAC operations required for a convolution layer. Our preliminary evaluation shows that not only can our Double MAC approach increase the *computation throughput* of a CNN layer by twice with essentially the same resource utilization, the network level performance can also be improved by 14∼84% over a highly optimized state-of-the-art accelerator solution depending on the CNN hyper-parameters.**

I. Introduction

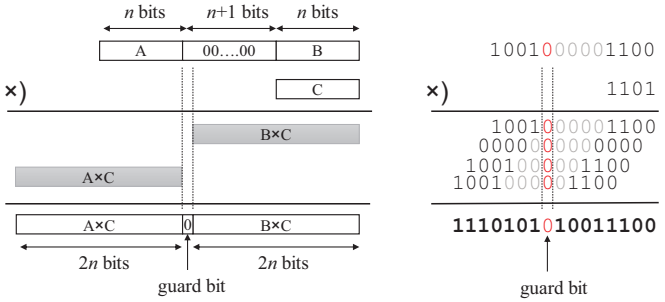
A unique option available only to hardware accelerators for convolutional neural networks (CNNs) [1]–[3] is the flexibility in arithmetic precision. In particular, FPGAs have the flexibility of choosing whatever precision sufficient for the target CNN application. In practice, however, FPGA mappings typically rely on DSP blocks for highest efficiency, and DSP blocks come with a fixed precision only. For instance, a DSP48E1 block in Xilinx FPGAs can perform a 25x18-bit multiplication followed by a 48-bit accumulation. Nonetheless, with some tweak we can pack two reduced-bitwidth multiplications into one DSP block, essentially doubling the computation rate of CNN algorithms than is currently possible with the best FPGA mapping techniques today.

This paper is about how to turn an ordinary DSP block of an off-the-shelf FPGA into a *Double MAC*, or a 2-way SIMD (Single-Instruction Multiple-Data) MAC (Multiply-and-Accumulate) unit, that can deliver 4 ops/cycle by performing two multiply-and-add operations simultaneously with reduced bitwidth. Though we evaluate our technique for a Xilinx Virtex-7 FPGA only, our method is generic and applicable to other FPGAs with similar hardware DSP blocks. Our technique does not require any change in the FPGA fabric itself.

Realizing SIMD addition on an FPGA is trivial and already supported. A SIMD MAC also is trivial if implemented with LUTs (Look-Up Tables). However the challenge is how to realize *SIMD multiplication on a DSP block* of an FPGA. Using DSP blocks is critical, since most CNN implementations on FPGAs rely on DSP blocks for MAC operations [1], and therefore being able to perform two MACs with one DSP block essentially means free 2X improvement in computation throughput. Further, though it is possible to increase throughput by other means (e.g., implementing additional MACs with LUTs [2]), our method is orthogonal to them.

Without native SIMD multiplication support on DSP blocks, we must create *virtual SIMD lanes* inside one DSP block, making sure that data on the lanes do not collide with each other. We must also take care of sign bits and overflow detection, all within one DSP block. Not only is this far from straightforward, a simple analysis reveals that a general SIMD multiplier on one DSP block is nearly impossible.

This research was supported by Nano·Material Technology Development Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT and Future Planning (2016M3A7B4909668).
*J. Lee is the corresponding author of this paper (E-mail: jlee@unist.ac.kr).



(a) n-bit SIMD unsigned multiplier (b) Example: 9×13 & 12×13

Fig. 1. How 2-way SIMD multiplication works (unsigned case).

In this paper we define a special class of SIMD multipliers, tailored for the kind of MAC operations found in convolutional layers of CNNs. Specifically, our Double MAC requires that the multiplications of a SIMD multiply share a common operand, viz., $A \times C$ and $B \times C$, and that the common operand, C, be an unsigned integer. We assume that only testing, as opposed to training, of a CNN is done on an FPGA. We also assume that the CNN accelerator accelerates convolution layers only, which account for the vast majority of computation.

We demonstrate that our Double MAC can double the computation throughput of a MAC array when given the same number of DSP blocks, as compared to the previous state of the art [1]. On the network level, i.e., with all convolution layers combined, our method can generate performance improvements that range, depending on the CNN hyper-parameters, from 14% to more than 80% over the highly optimized state-of-the-art accelerators, designed for two large, real-life CNNs [4], [5], without sacrificing the output quality by more than 1% (top-5 accuracy) after applying our scaling scheme.

II. Our Proposed Double MAC Architecture

A. SIMD Multiplication of Unsigned Numbers

Let us first consider the case where every operand is unsigned. Two unsigned multiplications with a common operand, i.e., $A \times C$ and $B \times C$, can be done using a single multiplier as illustrated in Figure 1. For this to work, two conditions must be met. First, the output register must be at least $4n$-bit wide for n-bit operands. Second, the two operands in one of the inputs must be separated by at least n bits.

While that is enough for a single multiplication, for the addition ensuing the multiplication to work without overflow, we need one *guard bit* as illustrated in the figure. In the accumulation mode, the same 1-bit guard bit can be used to *detect* any carry out of the lower $2n$ bits. On detection the carry is cleared immediately and the number of carry-out events is counted separately using a small counter. The counter value is added later once all the accumulation is done. Thus to perform two n-bit multiplications in a SIMD fashion we need one

978-3-9815370-8-6/17/$31.00 ©2017 IEEE

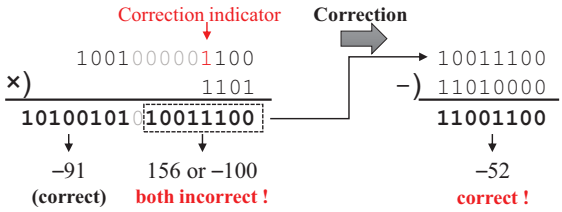


Fig. 2. Example SIMD execution: -7×13 and -4×13.

$(3n+1) \times n$-bit multiplier. For example, with the 25x18-bit multiplier of a DSP48E in Xilinx FPGAs, n can be at most 8.

B. SIMD Signed-Unsigned Multiplication

Let us consider how to perform a signed-unsigned multiplication using an unsigned multiplier. Let $B = b_{n-1} \cdots b_1 b_0$ be an n-bit signed integer, and C an n-bit unsigned integer. Recalling $-2^{n-1} = 2^{n-1} - 2^n$, the product $B \times C$ can be computed as follows, where $\hat{B}$ represents the value of B interpreted as an unsigned number.

$$\begin{aligned} B \times C &= (-b_n \cdot 2^{n-1} + \sum_{i=0}^{n-2} b_i \cdot 2^i) \cdot C \\ &= (b_n \cdot 2^{n-1} + \sum_{i=0}^{n-2} b_i \cdot 2^i) \cdot C - b_n \cdot 2^n \cdot C \\ &= \hat{B} \cdot C - b_n \cdot 2^n \cdot C \end{aligned}$$

In other words, $B \times C$ can be computed by performing an unsigned multiplication on B and C followed by a subtraction of n-bit left-shifted C if B is negative.

Extending this to 2-way SIMD multiplication is straightforward. We can perform two n-bit signed-unsigned multiplications in a SIMD fashion by performing one $(3n+1) \times n$-bit unsigned multiplication, followed by at most two subtractions.

However we can reduce the number of subtractions to one by performing a *signed* multiplication for the $(3n+1) \times n$-bit multiplication. In this case the upper $2n$-bit (i.e., $A \times C$) is correct (no need for correction), and only the lower $2n$-bit needs correction if B is negative, as demonstrated by example in Figure 2.

The correction term could be added in-place if we do just one multiply-add operation. In the accumulation (i.e., MAC) mode however we must delay doing corrections until the end of accumulation because we have only one guard bit. Instead the correction terms are accumulated separately in a small accumulator, whose value is later subtracted from the main accumulator.

C. Our Double MAC Architecture

Figure 3 illustrates our Double MAC architecture. One DSP block is needed to implement both a 2-way SIMD multiplier and a 2-way SIMD adder. And each of the SIMD multiplier and the SIMD adder has a correction output signal, which is shown as thin arrows in the figure. The correction outputs are accumulated into a separate register/counter and used later for final adjustment.

During adjustment there are two terms that need to be added/-subtracted. The adder-correction term C_1, which comes from the carry-out counter, has no overlap with the main accumulator output, and thus they can be just concatenated. Therefore we need only one addition, or the subtraction of the multiply-correction term C_2 from the concatenation result.

The width of the carry-out counter is determined from the number of values accumulated. In the case of our baseline CNN accelerator (see Section II-D), the width α must satisfy this constraint $\alpha \geq$

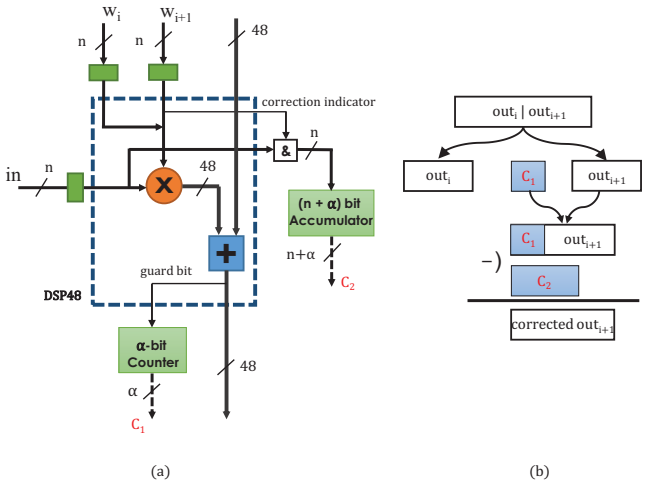


Fig. 3. (a) Datapath of our Double MAC architecture and (b) how it works.

$\log_2(K^2N/T_N)$ for every layer of the CNN, where K is the size of convolution filter in one dimension. Similarly the width of the accumulator for multiplier correction terms is $n + \alpha$.

The post-accumulation adjustment can be done easily by using an extra adder of $(n + \alpha)$-bit width that resides outside the DSP block. There is virtually no runtime overhead due to this adjustment, since the adjustment can be done simultaneously while new values are loaded into the MAC.

D. CNN Accelerator

A convolutional layer takes as input a number (N) of matrices called *input feature maps*, and generates a number (M) of matrices called *output feature maps*. The computation of each output feature map (Y_m) involves the summation of N 2D-convolutions between each of the input feature maps (X_n) and each of the N weight parameter matrices $(W_{m,n})$. Ignoring bias addition, $Y_m = \sum_n^N W_{m,n} * X_n$, where $*$ represents convolution.

Naturally the repetitions along the M, N dimensions have been the source of parallelism exploited by multiple hardware accelerators recently [1], [6]. In particular, a very recent CNN accelerator developed for FPGAs [1] is based on a 2D array of multipliers and adders This MAC array consumes T_N inputs and generates T_M outputs simultaneously, and performs $T_N T_M$ MAC operations per cycle sweeping through the entire input feature maps and output feature maps in a manner similar to loop tiling. Finding the best values for T_N and T_M can be done trivially using an exhaustive search, which may also consider other parameters such as buffer size parameters to maximize data reuse, since in general the input/output feature maps cannot fit in the on-chip memory of an FPGA.

Our proposed SIMD MAC architecture is applicable to other CNN accelerators as well. However, to demonstrate the applicability of our technique and evaluate performance improvement in the most realistic setting, we use the accelerator architecture of [1], with a small enhancement to maximize the compute density.

E. Maximizing Accuracy by Shift-only Scaling Scheme

One important concern for a reduced-precision scheme such as ours is lower accuracy at the network level. In this section we analyze the precision impact of our Double MAC architecture and present a very low-cost mitigation scheme.

Recall that all three operands of a Double MAC have 8-bit precision but the output has $(16 + \alpha)$-bit (lower word) and 31-bit $(= 48 - 17,$

upper word) precision, which should be enough for DNN applications. Also post-MAC steps such as max-pooling, normalization, and activation function, are independent of our Double MAC and therefore should not suffer from lower precision. Further, there is no loss of accuracy such as approximate addition *inside* our Double MAC architecture. Thus the only loss of accuracy by our architecture is caused by the truncation of input feature maps (including the primary input) and weight parameters.

One effective way to mitigate the loss of accuracy for such a case is to scale the input operands. In particular we pre-scale both weight parameters (including bias) and input feature maps, and later scale them back to their original ranges after all MAC operations are finished. The scaling factor can be set differently for differently layers. Let s_i and s_w be the scale factor for input feature maps and weight parameters, respectively, for a certain layer. Then $X' = s_i X$, $W' = s_w W$, and $b' = s_i s_w b$ (bias). Then the output of MAC can be scaled back by dividing it by $s_i s_w$. This transformation is exact, in that it does not change the result of computation when the precision is high enough, and when the precision is low, it can reduce accuracy loss due to input truncation. Further if s_i and s_w are each a power of 2, scaling can be nearly free; it may not be entirely free because scaling values can be different from layer to layer in which case we need a few muxes. Weight parameters and bias values can be scaled off-line.

Parameters s_i and s_w are per-layer parameters, whose optimal values are determined through profiling. We determine these parameters such that $P \cong 0.99$, where P is the probability of scaled values being within the range of 8-bit precision.

III. Experiments

A. Experimental Setup

To evaluate our technique we use two real-life CNN graphs: AlexNet [4] and VGG [5]. AlexNet consists of 8 layers, including 5 convolution layers. There are several configurations of VGG, in this paper we use the configuration that has 16 layers, including 13 convolution layers. Both networks use the ReLU activation function. Our baseline CNN accelerator is from [1] The accelerator performs the MAC operations only.

We have extended the Caffe framework [7] to obtain testbench data for RTL validation as well as to measure the output quality of limited-precision networks. To simulate the fixed-point behavior of the convolutional layer we have added a quantization layer before every convolutional layer. The output of convolutional layers is used to generate test vectors. The configuration for n-bit fixed-point is (n, 0), which means all n bits are dedicated for the integer part (including sign bit) and none is assigned for the fractional part, as we find that this configuration gives the best recognition performance. Our accuracy analysis is done with n ranging from 4 to 11.

For our performance comparison we set the clock frequency to 280MHz, which agrees with our synthesis results, and the memory bandwidth to 9 GB/s based on our RTL simulation.

B. Synthesis Results

Table I compares the resource requirements for one MAC unit for different precisions. The data for floating point and 32-bit fixed-point are taken from [1], and added for comparison.

We have implemented our SIMD MAC array in Verilog RTL and validated its functionality using Vivado simulator. The MAC array is synthesized using Vivado 2015.2 targeting the Xilinx Virtex7 485T FPGA, which has 2,800 DSP blocks. The MAC array has two key parameters, T_M and T_N, also called *tile parameters*, that impact the throughput of the accelerator. Their optimal values, shown in Table II, are found using exhaustive search for AlexNet. As expected, with the same level of DSP utilization our SIMD architecture can implement a MAC array twice as large as the non-SIMD version.

TABLE I
Resource requirements of different data precisions

Data type	DSP	LUT	FF
Floating-point 32-bit [1]	5	349	355
Fixed-point 32-bit [1]	4	0	0
Fixed-point 16-bit	1	0	0
Fixed-point 8-bit	1	0	0
Fixed-point 8-bit, Double MAC	0.5	11	12

TABLE III
Performance comparison - AlexNet (time unit: ms)

Layer	16b-fixed (T_1)	8b-fixed (T_2)	8b-fixed-D (T_3)	Runtime ratio T_1/T_3	Runtime ratio T_2/T_3
L1	1.31	1.31	1.31	1.00	1.00
L2	0.33	0.26	0.13	2.50	2.00
L3	0.13	0.13	0.08	1.54	1.54
L4	0.10	0.10	0.06	1.54	1.54
L5	0.10	0.07	0.04	2.23	1.49
Total (ms)	1.96	1.86	1.63	1.20	1.14
Power (W)	4.81	4.35	5.78	0.83	0.75
Energy (J)	9.43	8.10	9.42	1.00	0.86
(T_M, T_N)	(11, 192)	(32, 64)	(64, 64)		

The table also reports the area and maximum frequency of the synthesized circuit. Though our SIMD MAC array consumes more LUT and FF, which is due to the correction circuit, the additional resource usage is not high. On the other hand, being able to turn the extra resources into performance improvement can be an advantage, especially when those resources are typically underutilized in CNN accelerators unless a floating point data type is used (see the table).

C. System Performance Analysis

Table III and Table IV compare the run time of each convolutional layer of AlexNet and VGG, comparing three cases: 16-bit fixed-point, 8-bit fixed-point (baseline), and our 8-bit fixed-point SIMD implementations. We also show the total runtime, power and energy consumption for all the convolution layers. Both applications run on Virtex7 485T FPGA (the DSP count is 2,800 but limited by 80% usage limit). We use the optimal tile parameters for each case. The tile parameters suggest that our SIMD MAC array has exactly twice

TABLE IV
Performance comparison - VGG (time unit: ms)

Layer	16b-fixed (T_1)	8b-fixed (T_2)	8b-fixed-D (T_3)	Runtime ratio T_1/T_3	Runtime ratio T_2/T_3
L1	3.23	1.61	1.61	2.00	1.00
L2	3.23	3.23	1.61	2.00	2.00
L3	1.61	1.61	0.81	2.00	2.00
L4	3.23	3.23	1.61	2.00	2.00
L5	1.61	1.61	0.81	2.00	2.00
L6	3.23	3.23	1.61	2.00	2.00
L7	3.23	3.23	1.61	2.00	2.00
L8	1.51	1.61	0.81	1.88	2.00
L9	3.02	3.02	1.61	1.88	1.88
L10	3.02	3.02	1.61	1.88	1.88
L11	0.93	0.76	0.46	2.05	1.66
L12	0.93	0.76	0.46	2.05	1.66
L13	0.93	0.76	0.46	2.05	1.66
Total (ms)	29.71	27.67	15.07	1.97	1.84
Power (W)	5.93	5.81	7.85	0.75	0.74
Energy (J)	176.16	160.79	118.39	1.49	1.36
(T_M, T_N)	(64, 35)	(35, 64)	(64, 64)		

TABLE II
SYNTHESIS RESULTS FOR OPTIMAL TILE PARAMETERS

	AlexNet				VGG		
	32b-float [1]	16b-fixed	8b-fixed	8b-fixed-D	16b-fixed	8b-fixed	8b-fixed-D
Optimal (T_M, T_N)	(7, 64)	(11, 192)	(32, 64)	(64, 64)	(64, 35)	(35, 64)	(64, 64)
Max frequency (MHz)	280	280	280	280	280	280	280
DSP usage (%) (*)	80	75	73	73	80	80	73
LUT usage (%)	61.30	0.39	1.12	16.98	2.08	1.23	16.98
FF usage (%)	33.87	0.11	0.32	8.88	0.56	0.35	8.88
BRAM (KB)	4608	656	343	369	1295	1248	1632

*Note: DSP usage is limited to 80% as in the previous work [1].

the number of MAC units than that of the baseline (i.e., 8-bit fixed-point case). Thus it is possible to achieve 2× speedup in principle, as is often seen for VGG.

However in the first layer, the speedup (i.e., runtime ratio) over 8-bit fixed-point is none for either application. This is because not all the MAC units are always utilized due to the fragmentation issue, which is worsened especially when the tiles size is large compared to layer parameters. In the AlexNet, for instance, the first layer has layer parameters $M = 64$ and $N = 3$, but the tile parameters found are $T_M = 64$ and $T_N = 64$, which means that 61 out of 64 input ports are not utilized during the first layer. Because of this the first layer receives no benefit from the increased MAC array size of our SIMD MAC. To make matters worse L1 dominates the AlexNet's runtime, dragging down the overall speedup. VGG has the same issue in the first layer (having $M = 64, N = 3$) but the first layer has a low weight.

On the other hand, the first layer's performance can be changed by the choice of unroll dimensions (tile dimensions) in designing the MAC array. If one chooses, for instance, R and C loops [8], [9] instead of M and N, the first layer will scale much better. In fact R and C have highest values in the first layer, so doubled tile parameters may directly translate into increased performance.

Another point is that speedup varies layer by layer. This is because the tile parameters are sub-optimal for some layers, causing either under-utilization or fragmentation.

Regarding estimated power, since our method uses extra resources to implement the correction circuit, the power is higher than that of the baseline. In case of AlexNet, the runtime improvement our method achieves is not enough to compensate for higher power, resulting in higher energy consumption. However, in case of VGG, since our SIMD MAC array executes significantly faster than the baseline, it achieves about 33% energy reduction compared to the 8-bit fixed-point implementation.

D. Accuracy and Effect of Scaling

Figure 4 shows the output quality (i.e., our top-5 accuracy divided by that of the floating-point implementation) of our quantized version as we vary the precision from 4-bit through 11-bit, with and without our scaling optimization. We use the entire validation set of ILSVRC-2012.

For this experiment we run the entire network on Caffe, and fixed-point quantization is applied only in front of convolution layers (for input feature map and weight parameter data). Scale values are all powers of 2, so that free scaling-back is possible.

Our results reveal that despite their large size, both CNNs are surprisingly resilient up to 8-bit precision, when the quality starts to quickly deteriorate. Our scaling optimization can delay this by 1 or 2 bits. The quality degradation at 8-bits, which is the precision used in our SIMD evaluation, is less than 1%.

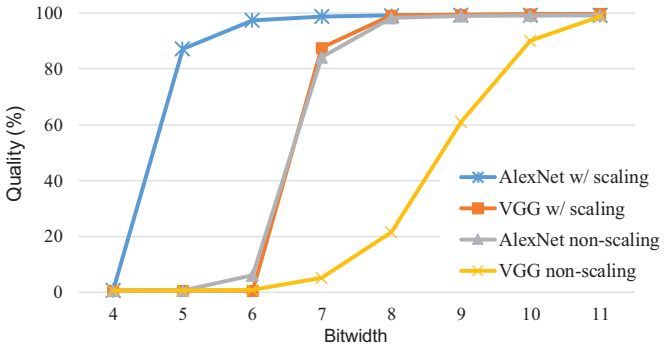


Fig. 4. Relative quality with various bit-widths of AlexNet and VGG, with and without scaling.

IV. CONCLUSION

We presented how to increase the computation rate of CNN accelerators on FPGAs by packing multiple MAC operations into one DSP blocks of off-the-shelf FPGAs. By exploiting the context in which MAC operations are used, our method can strike a good balance between usability and implementation overhead. Our experimental results demonstrate that our Double MAC can increase computation throughput of a CNN layer often by twice, and achieve signficant performance improvements on the network level ranging from 14% to more than 80% over an already optimized accelerator solution. Our scheme does use more LUTs, but LUTs are typically under-utilized in CNN accelerators, and our scheme can help improve balance in resource utilization. All these are done without sacrificing the output quality significantly. This improvement in performance can directly translate into energy saving, which can make accelerator solutions even more appealing as compared to GP-GPU solutions.

REFERENCES

[1] C. Zhang *et al.*, "Optimizing FPGA-based accelerator design for deep convolutional neural networks," in *FPGA '15*. ACM, 2015, pp. 161–170.
[2] J. Qiu *et al.*, "Going deeper with embedded FPGA platform for convolutional neural network," in *FPGA '16*. ACM, 2016, pp. 26–35.
[3] A. Rahman *et al.*, "Design space exploration of FPGA accelators for convolutional neural networks," in *DATE '17*, 2017.
[4] A. Krizhevsky *et al.*, "ImageNet classification with deep convolutional neural networks." in *NIPS*, P. L. Bartlett *et al.*, Eds., 2012, pp. 1106–1114.
[5] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," vol. abs/1409.1556, 2014.
[6] Y. Chen *et al.*, "DaDianNao: A machine-learning supercomputer," in *MICRO-47*. IEEE Computer Society, 2014, pp. 609–622.
[7] Y. Jia *et al.*, "Caffe: Convolutional architecture for fast feature embedding," *arXiv preprint arXiv:1408.5093*, 2014.
[8] C. Farabet *et al.*, "CNP: An FPGA-based processor for convolutional networks," in *FPL '09*, Aug 2009, pp. 32–37.
[9] A. Rahman *et al.*, "Efficient FPGA acceleration of convolutional neural networks using logical-3D compute array," in *DATE '16*, 2016, pp. 1393–1398.