Parallel Dot-Products for Deep Learning on FPGA

Mário Véstias
INESC-ID, ISEL,
Instituto Politécnico de Lisboa
mvestias@deetc.isel.pt

Rui Policarpo Duarte, José T. de Sousa, Horácio Neto
INESC-ID, Instituto Superior Técnico,
Universidade de Lisboa, Portugal
rui.duarte@tecnico.ulisboa.pt, jose.desousa@inesc-id.pt, hcn@inesc-id.pt

Abstract—**Deep neural networks have recently shown great results in a vast set of image applications. The associated deep learning models are computationally very demanding and, therefore, several hardware solutions have been proposed to accelerate their computation. FPGAs have recently shown very good performances for these kind of applications and so it is considered a promising platform to accelerate the execution of deep learning algorithms. A common operation in these algorithms is multiply-accumulate (MACC) that is used to calculate dot-products. Since many dot products can be calculated in parallel, as long as memory bandwidth is available, it is very important to implement this operation very efficiently to increase the density of MACC units in an FPGA. In this paper, we propose an implementation of parallel MACC units in FPGA for dot-product operations with very high performance/area ratios using a mix of DSP blocks and LUTs. We consider fixed-point representations with 8 bits of size, but the method can be applied to other bit widths. The method allows us to achieve TOPs performances, even for low cost FPGAs.**

Index Terms—**Multiply-accumulate, Deep learning, FPGA.**

I. Introduction

Dot product operation lies at the heart of many algorithms, including Convolutional Neural Networks (CNN). CNN are used successfully in a large number of applications, but demand more computational resources and memory bandwidth when compared to other methods. A huge number of parallel dot products are needed in such applications [1], [2], [3], [4]. These dot products are usually implemented using multiply-accumulate units with several degrees of parallelism.

To reduce the complexity of dot product operations in these networks, a few authors have shown that floating-point computation can be avoided for deep learning inference, while keeping the same accuracy of the results. In [5], the author develops and tests 8-bit approximation algorithms to increase memory bandwidth efficiency. The results show that these approximations maintains the quality of results on several networks with an increase in data transfer and consequently in computational performance, since it allows more parallelism.

Similar results were obtained by Gysel et al. [6]. In this work, several known complex CNN can be successfully implemented with 8-bits data, if 1% error compared to floating-point representation can be tolerated. This work provides a fine-tuned network approximations for well-known CNN using fixed-point representations (LeNet, Full CIFAR-10, SqueezeNet, CaffeNet, GoogleNet). In most cases, 8 bit fixed-point data representations are enough or even less.

Dot-products and, in particular, multiply-accumulate (MACC) can be implemented on FPGA using LUTs or DSP resources. When using a DSP to directly implement a single MACC of 8-bit operands is somehow inefficient since the DSP has a fixed internal MACC and so upper bits are just filled with 0s or 1s. Multiply-accumulate can also be implemented using the LUTs of the FPGA fabric. The operating frequencies are similar but, in this case, the unit is designed to exactly fit the operands size.

To better utilize the DSP resources, a recent white paper from Xilinx [7] have proposed a solution to implement two integer 8x8 MACCs in a single DSP (DSP48E2 [9] present in the Ultrascale families of FPGAs from Xilinx [8]). The proposed design accumulates $A \times C$ and $B \times C$, with two separate MACC units. The multiplier of the DSP is used to do both multiplications at the same time and the output adder accumulates both results independently. In fact, they only achieve a factor of 1.75 MACCs per DSP since an extra DSP is required to avoid accumulation overflow. This solution is restricted to second generation DSP slices and does not consider LUT fabric resources. Also, the paper does not explain how to separate the final accumulation results.

In this paper, we propose a design to implement multiple MACC units in parallel for dot-product calculation using DSPs (DSP48E1 or DSP48E2) and LUTs to achieve the maximum density of operations in a single FPGA. These designs can be used efficiently in many applications with intensive dot product calculation, including deep learning, matrix multiplication, etc., or other operations with parallel MACC operations. Specifically, the main contribution of this work is to design MACC units for dot-product using LUTs and DSP for 8-bit operands. Each DSP supports two multiplications.

The proposed parallel MACC units allow us to obtain parallel dot-product calculations with performances near one Tera operations per second (TOPs) in low cost FPGAs and above 10 TOPs for high density FPGAs. As far as we know, this is the first proposal of parallel MACC units with a mix of DSP and LUTs.

The rest of the paper is organized as follows. Section II describes the proposed design for parallel dot-products for 8-bit operands. Section III shows the results for both designs. Section IV concludes the paper.

II. Parallel dot-product for 8-bit operands

In this section, we describe our proposal for parallel dot-product calculation of 8-bit operands using both DSP and fabric LUTs in FPGA.

A. Dot-Product Parallelization

The dot product equation of two vectors, $X = (x_{n-1}x_{n-2} \dots x_1x_0)$ and $Y = (y_{n-1}y_{n-2} \dots y_1y_0)$, is well known and consists of a sum of multiplications:

$$X.Y = \sum_{i=0}^{i=n-1} x_i \times y_i \tag{1}$$

The operation can be implemented with a single MACC, where the multiplier calculates the product of vector elements with the same index which are then accumulated. The dot product of two vectors with *n* elements takes *n* cycles to calculate.

The dot product can be parallelized using several multipliers whose outputs are added and the result accumulated. For example, considering two multipliers, the dot product can be calculated according to the following expression:

$$X.Y = \sum_{i=0}^{i=n/2-1} (x_{2i} \times y_{2i} + x_{2i+1} \times y_{2i+1}) \tag{2}$$

A second method to parallelize the dot product calculation is to use several MACC units whose outputs are then added to obtain the final result. Formally, considering two MACC units, the dot product is calculated as follows:

$$X.Y = \sum_{i=0}^{i=n/2-1} (x_i \times y_i) + \sum_{i=n/2}^{i=n-1} (x_i \times y_i) \tag{3}$$

The second approach takes advantage of multiple MACC units but requires an output adder tree with adders of the same size of the accumulator. The former approach needs an accumulator separate from the multipliers but the addition of multiplier outputs is smaller. In out approach, we consider the first method, where the outputs of parallel multipliers are added before being accumulated.

B. Architecture of DSP Slices

DSP48E1 slice has one signed multiplier (25×18) and a 48-bits adder with a feedback loop to perform accumulation.

The DSP slice also contains a pre-adder whose output drives one input of the multiplier. This pre-adder is used to implement $(A + B) \times C$. This is not useful for a dot product operation but if we split the expression into $A \times C$ and $B \times C$ then we can calculate two separate dot products that have a factor (C) in common. This is common in many algebra operations, like vector- or matrix-matrix multiplications, and in deep learning algorithms.

The 48-bit adder of the DSP48E1 is in fact a ternary adder. However, since the multiplier generates two partial products to be added with the 48-bit adder, two of the inputs of the ternary adder are used for the multiplier (in the figure, we represent it with a single input). It means that there is only another input left for the adder that can be used to add a new input data, C in the figure, or to do accumulation.

In the DSP48E2 slices present in Ultrascale family of FPGAs from Xilinx, the adder has an extra input and the multiplier has been enlarged to 27×18.

With the extra input and the pre-adder, we can calculate $P = P + (X + Y) \times W + Z$.

In [7] they implement two MACC operations in a single DSP48E2. The two extra bits, compared to a DSP48E1 slice permit to accumulate up to seven product terms, before using the extra DSP. In a DSP48E1, only one product can be accumulated. So, the solution presented in [7] is inefficient for DSP48E1 based FPGAs.

C. Cell for 8-bit Dual Dot Product with DSP and LUT

The proposed cell supports the calculation of two independent dot products, DP0 and DP1, between a common vector, W, and vectors X and Y, $DP0 = X.W$ and $DP1 = Y.W$, in parallel, each using two multipliers. Formally, the dot products are calculated as follows:

$$DP0 = \sum_{i=0}^{i=n/2-1} (x_{2i} \times w_{2i} + x_{2i+1} \times w_{2i+1}) \tag{4}$$

$$DP1 = \sum_{i=0}^{i=n/2-1} (y_{2i} \times w_{2i} + y_{2i+1} \times w_{2i+1}) \tag{5}$$

The calculation of DP0 and DP1 is done iteratively over two registers, P0 and P1, using our cell according to equations:

$$P0 = P0 + x_{2i} \times w_{2i} + x_{2i+1} \times w_{2i+1}, i \in [0, \frac{n}{2} - 1] \tag{6}$$

$$P1 = P1 + y_{2i} \times w_{2i} + y_{2i+1} \times w_{2i+1}, i \in [0, \frac{n}{2} - 1] \tag{7}$$

The proposed cell for 8-bit operands calculates equations 6 and 7 with two independent multipliers implemented with a single DSP, two multipliers implemented with LUTs, uses the DSP adder to do two independent additions of products from DSP and LUTs, and performs two distinct accumulations outside the DSP using LUTs. This approach balances the utilization of LUTs and DSPs available in FPGAs.

Considering the input operands to DSP-based multipliers, X_{DSP}, Y_{DSP}, W_{DSP}, and the input operands to LUT-based multipliers, X_{LUT}, Y_{LUT}, W_{LUT}, the two accumulations, P_0 and P_1 are given by:

$$P0 = P0 + \underbrace{Y_{DSP} \times W_{DSP} + Y_{LUT} \times W_{LUT}}_{P_L} \tag{8}$$

$$P1 = P1 + \underbrace{X_{DSP} \times W_{DSP} + X_{LUT} \times W_{LUT}}_{P_H} \tag{9}$$

To implement the two 8-bit multiplications ($X_{DSP} \times W_{DSP}$ and $Y_{DSP} \times W_{DSP}$) with the single multiplier of a DSP slice, we separate both multiplicands, X_{DSP} and Y_{DSP}, so that the

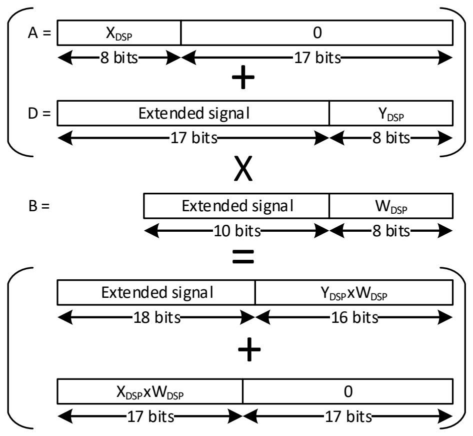

Fig. 1. Packing two 8-bit multiplications, $X_{DSP} \times W_{DSP}$, $Y_{DSP} \times W_{DSP}$, in a single DSP slice

products do not overlap, add them and then multiply by W_{DSP} (see figure 1).

Since the product of two 8-bit operands has 16 bits, eight bits are enough to separate them. However, we use nine bits. This allow us to add the result from the multiplier implemented with LUTs. The result of the operation illustrated in figure 1 is given by equation 11.

$$P = (X_{DSP}.2^{17} + Y_{DSP}) \times W_{DSP} \tag{10}$$

$$= \underbrace{X_{DSP} \times W_{DSP}}_{P_b}.2^{17} + \underbrace{Y_{DSP} \times W_{DSP}}_{P_a} \tag{11}$$

The $X.2^{17} + Y$ addition is performed by the pre-adder and the multiplication with W is performed by the multiplier of the same DSP.

From the result P, we have to extract the two products. As we can see from equation 11, the product $P_a = Y_{DSP} \times W_{DSP}$ is obtained directly from bits 15 downto 0 of P. Product $P_b = X \times W$ is obtained from bits 32 downto 17 of P when the the signal of $P_a$is positive, since in this case the bits relative to signal extension are zero. If product P_a is negative, then bits 32 downto 17 of P are the sum of $X_{DSP} \times W_{DSP} + 2^{16} - 1$ because of negative sign extension of the former multiplication. In this case, we must add 1 to the result to obtain the correct product. How this carry bit is generated, will be explained latter.

Both pairs of products are added using the 48-bit adder of the DSP according to the following equations.

$$P_L = Y_{DSP} \times W_{DSP} + Y_{LUT} \times W_{LUT} \tag{12}$$

$$P_H = X_{DSP} \times W_{DSP} + X_{LUT} \times W_{LUT} \tag{13}$$

where P_L is the sum of the least significant product from the DSP, P_a, with a product from a multiplier with LUTs and P_H is similar but with P_b. This is possible because the multiplicands in the DSP are separated by nine bits which allows one 8-bit addition.

The product P_a is always correct but the product P_b may have to be adjusted depending on the signal of P_a. If negative, a carry in must be added to P_b and, if positive, the result is already correct and must not be changed. We must consider this aspect when we add the products from LUTs. Considering the least significant products, $Y_{DSP} \times W_{DSP} = YW_{DSP}$ and $Y_{LUT} \times W_{LUT} = YW_{LUT}$, four different combinations of their signals are possible (see table I).

TABLE I
THE CORRECTNESS OF P_H ACCORDING TO THE SIGNALS OF THE LEAST SIGNIFICANT PRODUCTS FROM DSP AND LUTS

YW_{DSP}	YW_{LUT}	P_H
+	+	OK
+	-	??
-	+	??
-	-	OK

As we can see from the table, if both products are positive, then P_H is correct without any carry in. If both are negative, the carry in is automatically generated by the addition of both operands. When the products have different signs, the result of the addition may be negative or positive, and so P_H may be right or wrong, since a carry in may be generated when not required or not generated when required. To solve this, we manipulate the product implemented with LUTs in order to control the generation of the carry in, as follows:

- Least significant DSP product is positive: if the least significant LUT product is positive, we keep its sign, otherwise, we flip the bit signal. This guarantees that there is no carry propagation;
- Least significant DSP product is negative: if the least significant LUT product is negative, we keep its sign, otherwise, we flip the bit signal. This guarantees that there is carry propagation.

Finally, equations P_L and P_H must be independently accumulated.

$$P0 = P0 + P_L \tag{14}$$

$$P1 = P1 + P_H \tag{15}$$

Both accumulators are implemented with LUTs. Since we have changed the most significant bit of the LUT product to generate the correct carry in of P_H, the bit is recovered before being accumulated (see cell circuit in figure 2).

To improve the throughput, the circuit is pipelined (not represented in the figure). The multipliers implemented with LUTs have three levels of pipeline.

III. RESULTS

The proposed cell was implemented and tested in Xilinx's family 7 of FPGAs and SoC FPGAs. In all cases, we have characterized the cell in terms of area and performance, and determined the peak performance, that is, the performance that can be achieved by replicating the cell limited by the available resources of the FPGA.

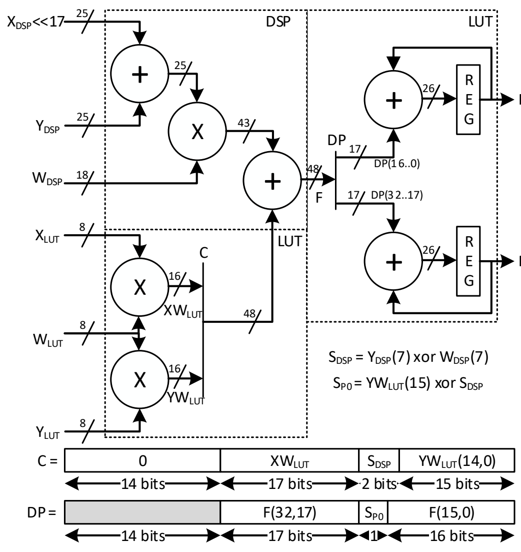

Fig. 2. Architecture of the proposed circuit for two parallel dot product accumulations using both DSPs and LUT)

TABLE II
TOTAL PERFORMANCE OF PARALLEL CELLS IN DIFFERENT FPGAS

FPGA	Total Resources			cell8		
	LUT	DSP	$\frac{LUT}{DSP}$	LUT	DSP	TOPs
ZYNQ7020	53200	220	242	39006	198	0,7
ZYNQ7045	218600	900	243	159570	810	2,7
XC7A35T	20800	90	231	15957	81	0,4
XC7A50T	32600	120	272	21276	108	0,5
XC7A200T	134600	740	182	131202	666	3,2
XC7K325T	203800	840	243	148932	756	3,6
XC7K355T	222600	1440	155	255312	1296	6,2
XCVX690T	433200	3600	120	388090	1970	10,4
XCVX980T	612000	3600	170	547660	2780	14,7
XCVX1140T	712000	3360	212	595728	3024	16,0

A. Characterization of the Cell

We first determined the resources required by a single cell. One 8-bit cell has 201 Flip-Flops, 197 LUTs and 1 DSP.

In terms of performance, we determined the operating frequency for different types of FPGAs from family-7 for speed grade -2 (see table III).

TABLE III
MAXIMUM OPERATING FREQUENCY (MHZ) ONE 8-BIT CELL

Type	cell8
Artix-7	422
Kintex-7	594
Virtex-7	660

The performance of the cell is close to the performance of a single DSP. The critical path when using Artix-7 is in the output of the accumulator. For the other two technologies, the critical path is in the DSP.

B. Peak Performances with Parallel Cells

We have determined theoretical peak performances assuming 90% occupation of the FPGAs with cells. No other logic is considered. Note, that this is a theoretical extrapolation since we are assuming the working frequency of a single cell. From these results, we can determine the performance of the system if a lower frequency is used.

Different FPGAs were considered, each with different LUT to DSP ratios (see table II).

As we can see from the table, it is possible to achieve performances close to one TOPs for low cost FPGAs and above 10 TOPs for high density FPGAs. Since FPGAs have different LUT to DSP ratio, the limiting resources may be DSPs or LUTs. For instance, the limiting resource for a XC7VX980T FPGA are the LUTs, while for the XC7VX1140T the DSPs are the limiting resources. A more balanced design is that having a LUT to DSP ratio close to the size of the cell.

IV. CONCLUSIONS

We have a 8-bit data cell for parallel dot product calculations. The cells are implemented with a mix of DSP and LUTs. The pipelined versions of the cells achieve very high throughput at frequencies from 450 MHz up to 650 MHz, depending on the FPGA family. The more balanced design is that having a LUT to DSP ratio close to the size of the cell. The 8-bit cell is now being applied in the design of a convolutional neural network for image recognition. With the proposed cell, we can achieve performances of TOPs and close to one TOPs in low cost FPGA based on Artix-7 technology.

ACKNOWLEDGMENT

This work was supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UID/CEC/50021/2013.

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, Imagenet classification with deep convolutional neural networks, in NIPS, 2012, pp. 10971105.
[2] M. D. Zeiler and R. Fergus, Visualizing and understanding convolutional networks, in ECCV, 2014, pp. 818833.
[3] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, Going deeper with convolutions, arXiv preprint arXiv:1409.4842, 2014.
[4] K. He, X. Zhang, S. Ren, and J. Sun, Deep residual learning for image recognition, arXiv preprint arXiv:1512.03385, 2015.
[5] T. Dettmers, "8-Bit Approximations for Parallelism in Deep Learning", International Conference on Learning Representations, 2016.
[6] T. Dettmers, "Hardware-Oriented Approximation of Convolutional Neural Networks", International Conference on Learning Representations, 2016.
[7] Y. Fu, E. Wu, A. Sirasao, S. Attia, K. Khan, and R. Wittig, "Deep Learning with INT8 Optimization on Xilinx Devices", White paper from Xilinx, WP486 (v1.0), November, 2016.
[8] Xilinx, "UltraScale Architecture and Product Data Sheet: Overview", Data Sheet DS890, February, 2017.
[9] Xilinx, "Ultrascale Architecture DSP Slice", User Guide UG569 (v1.3), November, 2015.