The Challenge of Multi-Operand Adders in CNNs on FPGAs: How Not to Solve It!

Kamel Abdelouahab, Maxime Pelcat, François Berry

The Challenge of Multi-Operand Adders in CNNs on FPGAs

How Not to Solve It!

Kamel Abdelouahab
Institut Pascal UMR CNRS 6602
Universite Clermont Auvergne
Clermont Ferrand, France

Maxime Pelcat
IETR UMR CNRS 6164, Institut Pascal
INSA Rennes
Rennes, France

Francois Berry
Institut Pascal UMR CNRS 6602
Universite Clermont Auvergne
Clermont Ferrand, France

ABSTRACT

Convolutional Neural Networks (CNNs) are computationally intensive algorithms that currently require dedicated hardware to be executed. In the case of FPGA-Based accelerators, we point-out in this work the challenge of Multi-Operand Adders (MOAs) and their high resource utilization in an FPGA implementation of a CNN. To address this challenge, two optimization strategies, that rely on time-multiplexing and approximate computing, are investigated. At first glance, the two strategies looked promising to reduce the footprint of a given architectural mapping, but when synthesized on the device, none of them gave the expected results. Experimental sections analyze the reasons of these unexpected results.

KEYWORDS

CNN, FPGA, Adder Trees, Approximate Computing

ACM Reference Format:
Kamel Abdelouahab, Maxime Pelcat, and Francois Berry. 2018. The Challenge of Multi-Operand Adders in CNNs on FPGAs: How Not to Solve It!. In *SAMOS XVIII: 2018 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation, July 15–19, 2018, Pythagorion, Samos Island, Greece.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3229631.3235024

1 INTRODUCTION

Since their breakthrough in 2012, Deep Convolutional Neural Networks (CNNs) [6] have become the *de-facto* standard used to solve an ever greater number of computer-vision tasks that range from image classification to semantic segmentation and scene recognition [3, 7, 13]. However, CNN-based algorithms are computationally intensive and their execution in real-time remains a challenging task, especially in embedded devices.

To address this challenge, a variety of dedicated accelerators, built around Field Programmable Gate Arrays (FPGAs) and Graphics Processing Units (GPUs), have been proposed. A key advantage of the former solution is its superior power efficiency when compared to the latter [10]. Moreover, CNN workloads have a streaming nature that is well suited to reconfigurable hardware architectures such as FPGAs, which motivated numerous research efforts to optimize FPGA implementation for CNNs [8, 11, 16]. Among the proposed methods, one possibility is to *directly map* a CNN graph on the FPGA resources, allocating each processing actor its own hardware instance, and each edge of the graph its own First-In First-Out (FIFO) channel [1].

In this paper, we point-out to a key feature of this Direct Hardware Mapping (DHM), which is the high hardware cost of *Multi-Operand-Adders*. More particularly, we found that 69% of the logic used to map the CNN graph on an FPGA is allocated to logic implementing *aggregated* adders which have the particularity to receive operands per thousands. To reduce these footprint of adders, we investigate in this work two strategies based on time-multiplexed serialization and approximate computing. Each method is promising on paper, but result in unexpectedly bad results when synthesized on FPGAs. Our experiments are reproducible and available on-line[1].

2 MULTI-OPERAND-ADDERS IN CNNS

A CNN graph takes the form of a succession of layers that hierarchically extract features from raw inputs. Most computation occurs in the *convolution* layers which rely on a learned set of N three-dimensional convolution filters of size $C \times J \times K$ to output a 3D feature map of size $N \times V \times U$. Thus, each filter involves a dot-product of $C \times J \times K$ elements as shown in equation 1.

$$\forall \{n, u, v\} \in [1, N] \times [1, V] \times [1, U]$$
$$Y[n, v, u] = \sum_{c=1}^{C} \sum_{j=1}^{J} \sum_{k=1}^{K} X[c, v + j, u + k].\Theta[n, c, j, k] \quad (1)$$

A method to accelerate the execution of these layers is to fully unroll the parallel computations involved in dot-products, and to map each multiplication to a dedicated hardware instance, as illustrated in Figure 1. In FPGAs, the advantage of this DHM strategy is to tile the circuitry of the multiplier according to the value of the multiplicand (i.e convolution filter) by applying Single Constant Multiplication (SCM) optimization techniques[14] where, for instance, multiplications by zero are removed and multiplications by a power of two are implemented by shift registers. As an example, a DHM-based implementation of the LeNet5 network requires ×8.6 less logic elements with this SCM optimization than without it, as detailed in [1].

A drawback of the DHM solution is that each layer requires N Multiple Operand Adders (MOAs) with $C \times J \times K$ inputs in order to accumulate the partial products. By default, synthesis tools

SAMOS XVIII, July 15–19, 2018, Pythagorion, Samos Island, Greece

ACM ISBN 978-1-4503-6494-2/18/07…$15.00
https://doi.org/10.1145/3229631.3235024

[1] https://github.com/KamelAbdelouahab/Multi-Operand-Adder

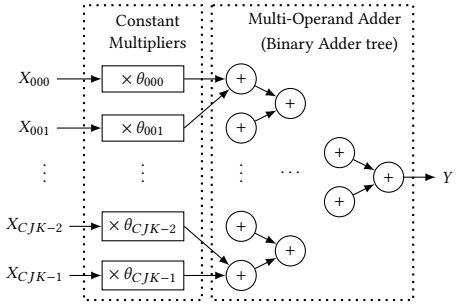

Figure 1: Direct Hardware Mapping of Dot Products in convolution layers: A Binary Adder tree sums the partial-products

instantiate deep binary adder trees[2] to implement MOAs, which require $CJK-1$ binary adders. However, for state-of-the-art CNNs, C,J,K can be large, leading to adders with up to 1774 operands (cf table 1.) As a result, most of the logic required to map a CNN layer is dedicated to the MOAs part, which corresponds, for instance, to 69% of the resources in the first layer of an AlexNet.

Table 1: Number of MOAs and number of mean non-null inputs per Adder in AlexNet layers

Layer	conv1	conv2	conv3	conv4	conv5
N	96	256	384	384	256
n_{opd}	325	957	1774	1398	1420

3 EXPLORED SOLUTIONS

In order to reduce the hardware resources instantiated during the mapping of a given convolutional layer, we investigate two strategies that are commonly used to reduce the footprint of MOAs. The first method iterates the accumulation of partial-sums through multiple clock cycles, leading to serialized adders. The second method relies on approximate computing techniques.

3.1 Serializing a cluster of adders

FPGA devices –and more particularly the Digital Signal Processing (DSP) blocks they embed– can run at a peak frequency that is much higher than the rate at which data and feature maps are acquired by a given CNN layer (~ 200 MHz for a DSP Block versus about 27.6 MHz for a 720p video stream). Given this, one can replace a cluster of binary adder trees by a serial accumulator that runs in a different, higher clock domain. In other words, we trade a clusters of n_c binary adders that previously operated at a frequency f_0 for a *single accumulator* that operates at a frequency f_c. In this context, $f_c = n_c f_0$ where $0 \leq n_c \leq n_{opd}$ and n_{opd} is the number of adder operands. In return, a parallel-to-serial register (serializer) is required to input the accumulator, as shown in Figure 2. In recent FPGA devices, this method can replace an $n_c \approx 6$-input MOA by a single accumulator and a pair of serializers, which may reduce the footprint of the MOA by a factor of $n_c - 1 \approx 5$ under the hypothesis that serializers have a simpler circuitry when compared to MOAs.

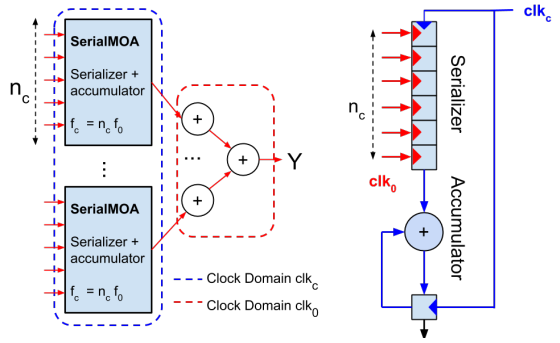

Figure 2: Architecture of a serial MOA. Each Serializer Accumulator Pair replaces a cluster of adders in the MOA of Figure 1

3.2 Approximate Adders

Deep CNNs are over-parametrized networks that tolerate by nature a degree of approximate computing. Approximations especially make sense during the inference phase because there is no error accumulation. Multiple state-of-the-art publications demonstrate the resiliency of CNNs towards compact bit-width arithmetic[2, 15] and even binarization[4, 12], which hints that CNNs may support others types of approximate computing techniques such as approximate adders. These adders, which use is limited to fault-tolerant applications, are known to deliver higher speed and power efficiency than exact operators [5].

In order to solve the challenge of MOA footprint reduction for CNNs, we leverage on the low resource utilization of the Lower-part-Or (LOA) approximate adders [9]. An LOA divides a b-bit adder into two sub-adders. The first one is an approximate l-bit sub-adder that computes the sum of least-significant bits by using a bit-wise OR operation. The second is an exact $(b-l)$-bit sub-adder that processes the most-significant bits using full adders. An extra AND gate is used to generate the carry-in signal for the exact adder part, as illustrated in Figure 3.

As pointed-out in the study of [5], LOA is the slowest but the most area efficient approximate adder, making it the best candidate for our study. In the Multi-Operand case, area saving may be achieved by replacing the exact binary adders in the tree with approximate adders such the LOAs.

4 EXPERIMENTS AND NEGATIVE RESULTS

4.1 Serialization

In order to study the impact of serialization on an MOA, we design[3] and synthesize the Serializer/accumulator pair of Figure 2. Figure 4 reports the logic utilization (in terms of Adaptive Logic Modules (ALMs)) of both the serializer, the accumulator and the serial adder for variable cluster sizes. These results are compared to the logic

[2]Binary adders refer to adders with TWO operands and NOT adders with a 1-bit operand

[3]Circuits are described in VHDL and synthesized on an Intel Stratix V 5SGXEA7 FPGA using Quartus 16.0. The bit-width of operands is 8 bits

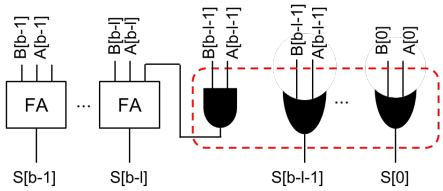

Figure 3: Hardware structure of a Lower-part OR approximate adder (LOA). Approximate parts in the red box. Each LOA Replaces a Binary Adder in the Tree of figure 1

utilization of the standard binary adder tree implementation of a MOA (in dashed line).

This figure shows a very unexpected result. The resources utilization of the serializer/accumulator pair exceeds the resources used by a fully pipelined implementation of an MOA (i.e a binary adder trees). This is the result of the costly logic fabric required by the serializer part, displayed in Figure 4, which grows linearly with the number of parallel inputs (i.e operands). The overhead of serializers thus invalidate the approach.

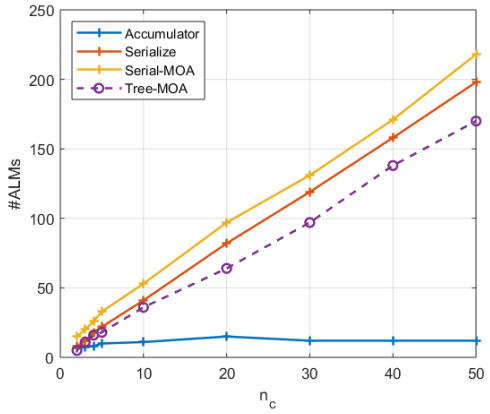

Figure 4: Comparison of the Logic resources used by a serialized and fully pipelined implementation of a MOA: The serializer results in a large resource overhead

4.2 Approximate Adders

In order to study the approximate LOA adder, we observe the impact of the *approximation ratio* on both the accuracy and hardware utilization of a binary adder. The approximation ratio is defined as the number of approximated bits per total bit-width l/b. A ratio of 0% corresponds to an exact adder while a ratio of 50% means that half of the bits of a given addition have been approximatively processed using OR gates.

To evaluate the accuracy of the method, the Mean Relative Error Distance (MRED) metric is used. Let $s = x + y$ be the result of an exact addition of x and y, and $\hat{s}$ the result of an approximate addition with same operands. The error distance is defined as:

$$MRED(s, \hat{s}) = mean\left(\frac{|\hat{s} - s|}{s}\right). \quad (2)$$

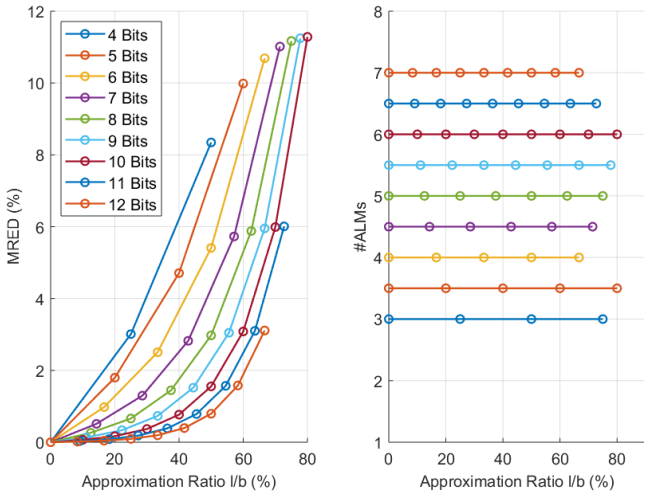

Figure 5: Error Rates and logic utilization of LOAs for variable bit-widths and approximation ratios.

The evolution of the MRED metric when varying bit-widths and approximation ratios is illustrated in Figure 5, as well as their corresponding logic utilization.

In terms of accuracy, using lower-part OR Adders results in a relatively small error (< 10% MRED for 8bits adders), which suggests that they might be exploited to derive energy-efficient CNN accelerators. However, in terms of hardware utilization, our experiments show that no area saving can be achieved on an FPGA when using LOAs. Indeed, the number of ALMs remains surprisingly constant, independently from the number of bits processed by an OR gate. This is explained by the fact that modern FPGA devices embed complex logical modules (ALM at Intel, Logical Blocks at Xilinx) that already contain a hard-wired full adder. This logical module either implements a full adder in the case of exact adders, or implements an OR gate in the case of approximate LOA adder. As a consequence, current FPGA and related hardware synthesizers do not benefit from approximate computing when targeting MOA adders and these results have been observed on both Intel and Xilinx FPGAs.

5 CONCLUSION

This paper has introduced the challenge of multi-operand adder footprint reduction when implementing a Convolutional Neural Network with direct hardware mapping on an FPGA. Two potential solutions have been studied, relying on serialization of adders and approximate computing. Though originally promising, these solutions have proven ineffective with current FPGA architectures that do not lend themselves well to adder approximation and serialization. The serialization of a cluster of adders does not reduce the footprint since the serializers require too many logic elements. The approximated adder is also ineffective, due to the structure of the logic blocks.

These conclusions motivate for introducing new specialized DSP blocks in FPGAs, implementing large adders fully in hardware.

6 ACKNOWLEDGMENT

This work was funded by the french ministry of higher education (MESR) and the LabEx IMobS^3 program at Institut Pascal (UMR 6602). We thank them and all the collaborators for their support to this research.

REFERENCES

[1] Kamel Abdelouahab, Maxime Pelcat, Jocelyn Serot, Cedric Bourrasset, and Francois Berry. 2017. Tactics to Directly Map CNN graphs on Embedded FPGAs. *IEEE Embedded Systems Letters* (2017), 1–4. https://doi.org/10.1109/LES.2017.2743247

[2] Suyog Gupta, Ankur Agrawal, Pritish Narayanan, Kailash Gopalakrishnan, and Pritish Narayanan. 2015. Deep Learning with Limited Numerical Precision. In *Proceedings of the International Conference on Machine Learning - ICML '15*. 1737–1746. http://jmlr.org/proceedings/papers/v37/gupta15.pdf

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition - CVPR '16*. 770–778. https://doi.org/10.1109/CVPR.2016.90

[4] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized neural networks. In *Advances in Neural Information Processing Systems - NIPS'16*. 4107–4115. http://arxiv.org/abs/1602.02830

[5] Honglan Jiang, Jie Han, and Fabrizio Lombardi. 2015. A Comparative Review and Evaluation of Approximate Adders. In *Proceedings of the Edition on Great Lakes Symposium on VLSI - GLSVLSI '15*. ACM Press, 343–348. https://doi.org/10.1145/2742060.2743760

[6] Alex Krizhevsky, Ilya Sutskever, Hinton Geoffrey E., and Geoffrey E Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems - NIPS'12*. 1âĂŞ9. https://doi.org/10.1016/j.protcy.2014.09.007

[7] Jonathan Long, Evan Shelhamer, and Trevor Darrell. 2015. Fully Convolutional Networks for Semantic Segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition - CVPR '15*. 3431–3440. https://people.eecs.berkeley.edu/~jonlong/long_shelhamer_fcn.pdf

[8] Yufei Ma, Naveen Suda, Yu Cao, Sarma Vrudhula, and Jae-sun Seo. 2018. ALAMO: FPGA acceleration of deep learning algorithms with a modularized RTL compiler. *Integration* (1 2018). https://doi.org/10.1016/j.vlsi.2017.12.009

[9] H R Mahdiani, A Ahmadi, S M Fakhraie, and C Lucas. 2010. Bio-Inspired Imprecise Computational Blocks for Efficient VLSI Implementation of Soft-Computing Applications. *IEEE Transactions on Circuits and Systems I: Regular Papers* 57, 4 (4 2010), 850–862. https://doi.org/10.1109/TCSI.2009.2027626

[10] Eriko Nurvitadhi, Suchit Subhaschandra, Guy Boudoukh, Ganesh Venkatesh, Jaewoong Sim, Debbie Marr, Randy Huang, Jason OngGeeHock, Yeong Tat Liew, Krishnan Srivatsan, and Duncan Moss. 2017. Can FPGAs Beat GPUs in Accelerating Next-Generation Deep Neural Networks?. In *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays - FPGA '17*. 5–14. https://doi.org/10.1145/3020078.3021740

[11] Jiantao Qiu, Jie Wang, Song Yao, Kaiyuan Guo, Boxun Li, Erjin Zhou, Jincheng Yu, Tianqi Tang, Ningyi Xu, Sen Song, Yu Wang, and Huazhong Yang. 2016. Going Deeper with Embedded FPGA Platform for Convolutional Neural Network. In *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays - FPGA '16*. ACM, New York, NY, USA, 26–35. https://doi.org/10.1145/2847263.2847265

[12] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. 2016. XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks. In *Proceedings of the European Conference on Computer Vision - ECCV'16*. https://arxiv.org/pdf/1603.05279.pdf

[13] Joseph Redmon and Ali Farhadi. 2018. *YOLOv3: An Incremental Improvement*. Technical Report. http://arxiv.org/abs/1804.02767

[14] Yevgen Voronenko and Markus Püschel. 2007. Multiplierless multiple constant multiplication. *ACM Transactions on Algorithms* 3, 2 (5 2007), 11–es. https://doi.org/10.1145/1240233.1240234

[15] Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. 2016. Quantized Convolutional Neural Networks for Mobile Devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition - CVPR '16*. 4820–4828. http://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/Wu_Quantized_Convolutional_Neural_CVPR_2016_paper.pdf

[16] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. 2015. Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks. In *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays - FPGA '15 (FPGA)*. 161–170. https://doi.org/10.1145/2684746.2689060