BNNsplit: Binarized Neural Networks for embedded distributed FPGA-based computing systems

Giorgia Fiscaletti, Marco Speziali, Luca Stornaiuolo, Marco D. Santambrogio, Donatella Sciuto
Politecnico di Milano, Dipartimento di Elettronica Informazione e Bioingegneria (DEIB), Milan, Italy
{giorgia.fiscaletti, marco.speziali}@mail.polimi.it
{luca.stornaiuolo, marco.santambrogio, donatella.sciuto}@polimi.it

***Abstract*—In the past few years, Convolutional Neural Networks (CNNs) have seen a massive improvement, outperforming other visual recognition algorithms. Since they are playing an increasingly important role in fields such as face recognition, augmented reality or autonomous driving, there is the growing need for a fast and efficient system to perform the redundant and heavy computations of CNNs. This trend led researchers towards heterogeneous systems provided with hardware accelerators, such as GPUs and FPGAs. The vast majority of CNNs is implemented with floating-point parameters and operations, but from research, it has emerged that high classification accuracy can be obtained also by reducing the floating-point activations and weights to binary values. This context is well suitable for FPGAs, that are known to stand out in terms of performance when dealing with binary operations, as demonstrated in FINN, the state-of-the-art framework for building Binarized Neural Network (BNN) accelerators on FPGAs. In this paper, we propose a framework that extends FINN to a distributed scenario, enabling BNNs implementation on embedded multi-FPGA systems.**

***Index Terms*—Binarized Neural Networks, BNN, PYNQ, embedded, distributed**

I. INTRODUCTION

With the huge growth in the use of Convolutional Neural Networks (CNNs), the need for increasingly efficient solutions in terms of performance and power consumption has developed in recent years. It is well known that CNNs, especially deeper networks, require a significant amount of resources to store all the parameters needed for the computations, i.e. weights and activations, and massive computational power. A solution to lighten computations and reduce power consumption and memory footprint has been shown to be the use of Binarized Neural Networks (BNNs) [1], [2]. In these networks, all the heavy floating point computations are mapped to low-precision arithmetic operations and can be reduced to binary operations. This is where Field Programmable Gate Arrays (FPGAs) come in handy, as noted by Zhou et al. [3] in their work on neural networks with low bitwidth parameters.

However, it is possible to achieve greater performance and a significant reduction of both the global resource usage the dynamic power consumption by leveraging the possibilities offered by distributing the computations on a multi-FPGAs system. This also leads to an improvement of the scalability of the accelerators, since it allows to perform computations on very deep neural networks whose parameters would not fit on the On Chip Memory (OCM) of a single board.

In this work we propose BNNsplit, an extension of the framework developed by Umuroglu et al. [4] for implementing BNNs inference accelerators on embedded distributed FPGA-based computing systems, focusing on the implementation of a binarized deep convolutional neural network on a multi-PYNQ-Z1 system.

II. BACKGROUND

The key reasons behind the choice of Field Programmable Gate Arrays (FPGAs) for this work are the low latency of the board and the high level of exploitable parallelism to perform a large number of convolutions. Moreover, FPGAs are one of the best hardware choices when dealing with binary values, as shown in [1], [2]. Some implementations of low-precision CNNs accelerators on FPGAs have already been developed, such as the work by Moss et al. [5] that presents a high performance BNN accelerator on the Intel®Xeon+FPGA™ platform and FINN [4], a framework developed by Xilinx that provides fast inference accelerators using a dataflow-style streaming architecture on PYNQ-Z1 and Ultra96 boards. More in detail, the authors of FINN described a novel set of optimizations to efficiently map the different layers of BNNs to the FPGA resources while maximizing the throughput requirements defined by the user. However, FINN assumes that all BNN parameters can fit into the available OCM of a single FPGA and, to the best of our knowledge, no solutions for BNNs on multi-FPGAs systems have ever been explored.

We started from this point to propose a methodology that extends the FINN framework in a distributed context. The major problem was to find an efficient way to divide one of the proposed neural networks without any performance loss. One of the core ideas of the network splitting was to reduce the resource usage, to let the framework handle larger neural networks with a greater amount of parameters (i.e. weights and activations) that would otherwise saturate the memory of a single board. In addition, another fundamental purpose of an efficient network division is to increase the throughput with respect to the single node, assuming that the given distributed system is not network-bounded. This could lead to lower resource usage (with respect to replicating the whole network on multiple parallel boards), allowing to exploit further procedures such as partial reconfiguration, to use different applications on the same system without loss in terms of performance.

978-3-9819263-4-7/DATE20/©2020 EDAA

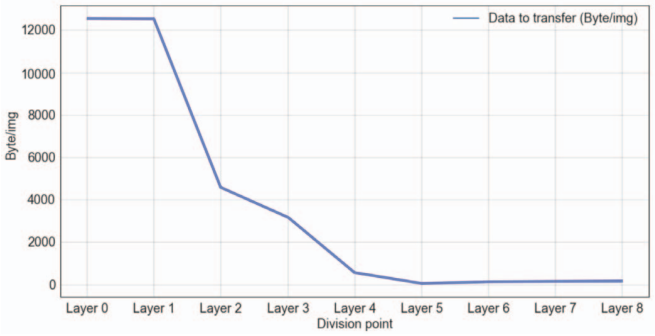

Fig. 1. Plot showing the amount of bytes per image to transfer at each layer.

III. Related Works

The idea of implementing BNNs on FPGA-based systems is not new. [6], [7] shows how FPGAs can provide orders of magnitudes in efficiency improvements over CPU/GPU-based solution. The authors implemented a BNN accelerator targeting a high-end Altera Aria 10 FPGA. In [8], authors demonstrate the advantages of using BNNs over classical CNNs solutions, implementing an FPGA-based accelerator that is synthesized from C++ to FPGA-targeted Verilog. [9] describes an accelerator for BNNs targeting the Maxeler MPC-X2000 platform, that shows a promising speed with respect to the low energy consumption. However, none of these solutions provides details on how to manage the ever-increasing size of BNNs architectures by exploiting distributed systems with multiple processing elements. [10], [11] demonstrates the advantages of using an FPGA-based distributed system, implementing both a framework for the efficient deployment of data analytics on a multi-PYNQ system mapping the Spark framework on FPGAs and an interface that is implemented with Remote Procedure Call to partition hardware design on multiple boards, through an Ethernet network. Yet, to date, [12] is the only work that shows an implementation of neural networks in a distributed FPGA system. The authors propose a CNN-based face detection algorithm implemented on an embedded, distributed system of multiple FPGAs. However, the approach is specific to the face detection context and uses quantization instead of binarization to optimize the neural network parameters.

IV. Methodology

1) BNN topology: The choice of the BNN to divide was based on both the size of the network and the type of the parameters. Since all the networks implemented by the FINN [4] framework fit the OCM of a single FPGA, the most significant BNN to split would certainly be the largest one. Furthermore, the proposed BNNs differ from each other in the type of parameters (i.e. weight and activations), that can be both 1 and 2-bit parameters. 2-bit parameters are more versatile during the HLS synthesis since the Multiply-Accumulate (MAC) operations performed on 2 bits can be implemented both on Mul LUT and DSP48 cores - while MAC operations on 1-bit parameters are simple XOR operations, not supported by the above-mentioned cores. Thereby, the chosen BNN is the CnvW2A2, a deep convolutional neural network with 6 convolutional layers and 3 fully connected layers, whose activations and weights are 2-bit parameters.

2) Splitting strategy: Since the objective of this work is to find an efficient way to distribute the framework on a multi-FPGAs system, a first strategy is to split the BNN so as to minimize the amount of data transferred between the PYNQs, building a pipeline for data processing. This option could allow avoiding network bottlenecks while reducing the resource utilization per board with respect to the full network. Additionally, the BNN parts can be replicated and reused in parallel in order to increase the overall throughput. Further concerns of the BNN splitting could be searching for a trade-off between the resource balance among the PYNQs and the amount of data transferred in the network, dividing the BNN into more than 2 parts.

3) Proposed design: The proposed BNN splitting was performed by searching for the layer whose output data was the smallest in terms of bytes per image. The analysis and evaluation of the possible divisions showed that the output data of the convolutional layers progressively decrease from 12544 Bytes/img after the first convolutional layer to just 64 Bytes/img after the last convolutional layer. The plot in Fig. 1 shows the decrease of bytes per image at different split points, from more than 12000 Bytes/img dividing after Layer 0 to 64 Bytes/img dividing after Layer 5. The selected BNN was therefore divided into two parts, choosing the split point between the last convolutional layer and the first fully connected layer. The first part (Part_1) handles the convolution and max pooling operations, while the second part (Part_2) performs the fully connected computations. The two resulted kernels process memory batches of `ap_uint<64>` via AXI4 ports, that are converted into streams at the beginning of the computation, see Fig. 2.

4) Resource optimization: In order to evaluate all the possible solutions in terms of resources utilization, three possible implementations of the MAC operations were explored - presented in the FINN [4] framework:

- **Default implementation:** this implementation of the multiplications leaves the choice of the resource to Vivado, that will automatically map the function to the hardware.
- **Implementation on DSP48:** this implementation force the use of DSP48 cores (`#pragma HLS resource core=DSP48`). However, the results obtained in terms of resource utilization is very similar to the Default implementation and the design fails to meet the timing requirements after Vivado synthesis.
- **Implementation on Mul_LUT:** this implementation exploits Multiplier implemented with LUTs (`#pragma HLS RESOURCE core=Mul_LUT`) and it is the best in terms of resources utilization and clock frequency. In fact, only a few DSP48 cores are used, reducing the power consumption by 13% with respect to the other two options.

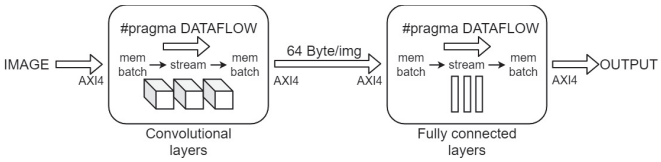

Fig. 2. Representation of the proposed design.

V. Evaluation

1) Resource utilization: The results of resource utilization obtained after the place and route phase on the convolutional part (Part_1) show that the best solution is to force the multiplications of the MAC on Mul_LUT cores. With this configuration, it is possible to reach a modest reduction of most of the resource utilization (-12% of LUT, -14.5% of LUTRAM, a slight reduction of FF and -11% of BRAM) with respect to the full BNN. Both the DSP48 and the default implementations have a slightly higher resource usage with respect to the Mul_LUT solution, and a massive - but expected - utilization of DSP48 for the multiplications, reaching the saturation of the available cores. This clearly leads to the higher power consumption of the DSP48 and default configurations (nearly +13% of power consumption with respect to both the full BNN and the convolutional block implemented with Mul_LUT). The fully connected part (Part_2) instead does not experience a significant difference between the different configurations.

2) Performance: The measured execution times and throughputs of the three different implementations were surprisingly very similar, probably due to the fact that the allocated DSP48 cores in both the default and the DSP48 alternatives are not actually needed at runtime. The fully connected block, being very light in terms of resources exploited and computation, and receiving a reduced amount of data in input due to the convolution operations previously performed on the image, is able to reach a potential data processing throughput of nearly 720 chunks/s - assuming no network latency for the time being. The convolutional part is instead much slower, due to the heavy computations performed within the layers. Nonetheless, its throughput is nearly 40% higher with respect to the full network on a single board. Both the mean and standard deviation of throughput and the execution time shown in Table I were computed over 30 executions for each configuration, reconfiguring a single PYNQ-Z1.

3) System distribution: With the aim of distributing the framework on a multi-FPGAs system, one must take into account the limits imposed by the network connection between the PYNQ-Z1 boards, that could significantly reduce the amount of data transferred per second, both between the boards and between boards and hosts. A potential solution to overcome the network problem could be to leverage the Gigabit Ethernet PHY of the PYNQ boards, making them communicate throughout a 1 Gbps connection. In this way, we could be able to transmit 125 MB/s, namely 312 img/s of 400 KB each. This means that the bottleneck would pass from the network side to the processing side, making the distributed

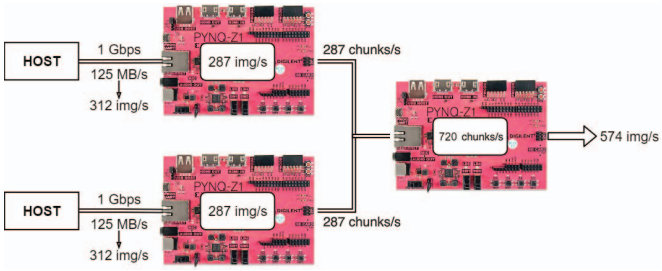

Fig. 3. The 3-PYNQ solution.

solution on two PYNQ-Z1 boards potentially competitive with the full BNN loaded on a single PYNQ-Z1. Since the network capacity is higher than the board throughput, this will result in a bottleneck on the processing side, lowering the potential amount of data transferred per second from 312 img/s to 205 img/s in the single board case - i.e. the full network throughput, see Table I. The two PYNQ-Z1 distributed solution instead would potentially allow exploiting the higher throughput of the convolutional layers, increasing the overall throughput by 40% with respect to a single board solution. In this network unbounded context, by replicating the full BNN on two separate PYNQs and parallelizing the two different computations, the result would simply be the double of the throughput of the single PYNQ, since the processing is limited by the PYNQ's throughput - i.e. 410 img/s.

By means of a 3-PYNQ distributed system (Fig. 3), one could not only leverage the network capacity but also exploit the 720 chunks/s - i.e. 64 Byte img/s - throughput of the fully connected block loaded on the third PYNQ. The amount of data per second arriving from the convolutional blocks would optimistically be the sum of the two convolutional throughputs, hence 574 img/s. Assuming for the time being a minor latency of the input buffer of the third PYNQ, the overall throughput could reach a maximum of 574 img/s, again a 40% throughput gain with respect to the full BNN on two separate PYNQs.

With the solution shown in Fig. 3 we would be able to achieve the 40% throughput gain not only with a minor raise of BRAM and FF usage (+3.6% BRAM, +4.4% FF), but also with a slight reduction of the overall LUT and LUTRAM utilization (-7.8% LUT, -27.41% LUTRAM) with respect to the system composed by 2 separate PYNQs replicating the full BNN. Moreover, if we consider a further system composed by 3 separate PYNQs with the full BNN, the net dynamic power consumption caused by the resource usage would increase by 39%, while the throughput gain with respect to the solution in Fig. 3 would be only 7% at most (651 img/s against 574 img/s). This makes the 3-PYNQ-Z1 system with just the replication of the convolutional block the best compromise between performance and power consumption. Nonetheless, a further increment in the number of boards of the distributed system would lead to the saturation of the fully connected block, that cannot support more than 720 chunks/s. With three replicas of the convolutional block, the throughput gain goes down to 10% with respect to three separate PYNQs in parallel performing the computations of the full BNN (the respective

TABLE I
RESOURCE UTILIZATION, POWER CONSUMPTION (P), THROUGHPUT (MEAN & STD) $[img/s]$, EXECUTION TIME (MEAN & STD) $[us]$

Part	Core	LUT %	LUTRAM %	FF %	BRAM %	DSP48 %	P $[W]$	Thr_m	Thr_std	ExT_m	ExT_std
Full	Mul_LUT	70.12	42.74	48.30	100.00	14.55	1.836	205.51	0.07	4867.77	1.36
Part_1	Mul_LUT	58.35	28.17	44.62	88.93	14.55	1.823	287.10	1.24	3480.23	1.63
Part_1	DSP48	63.58	28.28	46.34	88.93	100.00	2.106	287.22	0.44	3480.56	1.09
Part_1	Default	63.54	28.26	46.34	88.93	100.00	2.065	287.16	0.84	3480.40	1.36
Part_2	Mul_LUT	15.70	2.36	11.76	25.71	0.00	1.511	719.60	0.60	1389.70	1.19
Part_2	DSP48	15.79	2.41	11.69	25.71	4.09	1.519	719.52	0.57	1390.63	1.05
Part_2	Default	15.70	2.36	11.76	25.71	0.00	1.511	719.56	0.59	1390.17	1.12

overall throughputs are 720 chunks/s and 615 img/s). With 4 replicas of the convolutional blocks, all the advantages are lost.

4) Testing: To test our distributed solution, we took advantage of the FARD [13] framework, an event-based system that we modified to support jobs and pipelines. In particular, a job has been implemented as a task accepting data from an input queue that is killed after all the data has been processed. The pipeline has been created by generating at runtime the graph of peers (representing the pipeline) and sending configuration messages to start the jobs and to configure the bitstreams. The measurements in Table I show throughput and execution times of the single nodes, without considering communication overhead. The fully connected block, being very light in terms of resources exploited and computation, and receiving a reduced amount of data in input due to the convolution operations previously performed on the image, is able to reach a potential data processing throughput of nearly 720 chunks/s - assuming no network latency for the time being. The convolutional part is instead much slower, due to the heavy computations performed within the layers. Nonetheless, its throughput is nearly 40% higher with respect to the full network on a single board. The experimental testing, on a sample of 30 runs (each classifies 20 images), showed an average response time of 0.083767s with a standard deviation of 0.007515s.

VI. CONCLUSION

As demonstrated with this work, an efficient division of a BNN can actually lead to a potential enhancement of performances, making it the first step towards the extension of the FINN framework on a multi-FPGAs architecture. We presented a network divided into two distinct blocks, minimizing the data transferred between the boards. As a future direction, other splitting strategies, such as dividing the BNN into multiple partitions in order to extract higher performances and balance the resource utilization on the various boards, could be explored. The presented configuration with the network divided in convolutional block and fully connected block leaves a large amount of free resources on the board where the fully connected kernel is placed. Hence, future works could regard the leveraging of the free area to implement other applications without performance loss. For example, the free resources could be exploited by means of partial reconfiguration, i.e. modifying at runtime a portion of the bitstream loaded on the board.

REFERENCES

[1] M. Courbariaux and Y. Bengio, "Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1," *CoRR*, vol. abs/1602.02830, 2016. [Online]. Available: http://arxiv.org/abs/1602.02830

[2] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *Computer Vision – ECCV 2016*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds. Cham: Springer International Publishing, 2016, pp. 525–542.

[3] S. Zhou, Z. Ni, X. Zhou, H. Wen, Y. Wu, and Y. Zou, "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients," *CoRR*, vol. abs/1606.06160, 2016. [Online]. Available: http://arxiv.org/abs/1606.06160

[4] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "Finn: A framework for fast, scalable binarized neural network inference," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '17. ACM, 2017, pp. 65–74.

[5] D. J. M. Moss, E. Nurvitadhi, J. Sim, A. Mishra, D. Marr, S. Subhaschandra, and P. H. W. Leong, "High performance binary neural networks on the xeon+fpga™ platform," in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, Sep. 2017, pp. 1–4.

[6] E. Nurvitadhi, D. Sheffield, J. Sim, A. Mishra, G. Venkatesh, and D. Marr, "Accelerating binarized neural networks: Comparison of fpga, cpu, gpu, and asic," in *2016 International Conference on Field-Programmable Technology (FPT)*. IEEE, 2016, pp. 77–84.

[7] E. Nurvitadhi, G. Venkatesh, J. Sim, D. Marr, R. Huang, J. Ong Gee Hock, Y. T. Liew, K. Srivatsan, D. Moss, S. Subhaschandra *et al.*, "Can fpgas beat gpus in accelerating next-generation deep neural networks?" in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2017, pp. 5–14.

[8] R. Zhao, W. Song, W. Zhang, T. Xing, J.-H. Lin, M. Srivastava, R. Gupta, and Z. Zhang, "Accelerating binarized convolutional neural networks with software-programmable fpgas," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2017, pp. 15–24.

[9] S. Liang, S. Yin, L. Liu, W. Luk, and S. Wei, "Fp-bnn: Binarized neural network on fpga," *Neurocomputing*, vol. 275, pp. 1072–1086, 2018.

[10] E. Koromilas, I. Stamelos, C. Kachris, and D. Soudris, "Spark acceleration on fpgas: A use case on machine learning in pynq," in *2017 6th International Conference on Modern Circuits and Systems Technologies (MOCAST)*. IEEE, 2017, pp. 1–4.

[11] E. Tapia-Morales, J. Martínez-Carballido, and G. Castro-Muñoz, "Scheme for partitioned co-emulation using multi pynq-z1 boards," in *Proceedings of the International Conference on Embedded Systems, Cyber-physical Systems, and Applications (ESCS)*. The Steering Committee of The World Congress in Computer Science, Computer ..., 2018, pp. 17–21.

[12] A. M. Nestorov, A. Scolari, E. Reggiani, L. Stornaiuolo, and M. D. Santambrogio, "A case study for an accelerated dcnn on fpga-based embedded distributed system," in *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 2019, pp. 91–94.

[13] S. Barbieri, F. Casasopra, R. Brondolin, and M. D. Santambrogio, "Fog acceleration through reconfigurable devices," in *Proceedings of the Research and Technologies for Society and Industry (RTSI), 2019 IEEE*, ser. IEEE 5th International Forum. IEEE, 2019, pp. 138–143.