Hardware resources analysis of BNNs splitting for FARD-based multi-FPGAs Distributed 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*—FPGAs have proven to be valid architectures to accelerate the inference phase of Convolutional Neural Networks (CNNs). State-of-the-art works also demonstrated that it is possible to take advantage of a distributed FPGA-base system to improve performance, power consumption and scalability of such algorithms. However, the hardware resource usage, communication, and the nodes management become main aspects when dealing with an embedded distributed scenario. In this context, FINN optimizes the FPGA-based CNNs with binarization and FARD is a framework that allows the acceleration of fog computing-based application with FPGAs. In this work, we present how to extend FARD to deal with job-based applications rather than the event-based fog computing scenario. In particular, we analyzed two PYNQ-Z1 connected each other and we implemented a distributed BNN algorithm based on FINN's CnvW2A2. Results show how hardware resources vary according to the division of the network when splitting after each convolutional layer.**

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

I. Introduction

The last decade saw an exponential growth of convolutional neural networks (CNN) as one of the most exploited computational models in fields such as image recognition, natural language processing, anomaly detection, and so on. Alongside the development of convolutional neural networks and their newly discovered applications, there is a growing need for infrastructures that are capable to perform the complex calculations required in this computational model. Also, these new systems need to be able to provide a large amount of memory - necessary to process and transfer the huge volume of data that flows throughout the network, as well as the required parameters - and computational power. A solution to lighten computations and significantly reduce power consumption and memory footprint has been shown to be the use of Binarized Neural Networks (BNNs) [1]. It has been demonstrated in different works - i.e. [1]–[3] - that it is possible to quantize and reduce the precision of weights and activations with just a minimum loss of accuracy. This way, all the heavy floating-point computations can be easily mapped to low-precision arithmetic operations. In particular, almost all the computations performed in BNNs can be reduced to binary operations. This is where FPGAs come in handy, as noted by Zhou et al. [2] in their work on neural networks with low bitwidth parameters. FPGAs are well known for being very suitable to achieve great performance when dealing with logic and fixed precision values, and therefore with binary values.

In this work, we propose an analysis of the hardware resource usage while modifying the CNN splitting point and we describe how FARD [4], a framework to implement fog computing distributed system accelerators, is modified to deal with this kind of applications. This is a continuation of our previous work BNNsplit [5], where we explored splitting strategies for Xilinx FINN BNNs [6]. With respect to the previous work, where the division of the network was performed in order to minimize the amount of data that is sent from one layer to another, now it takes into consideration balancing the amount of hardware resources on each node.

The rest of this paper is organized as follows: section II presents the background, including the description of FARD and of the chosen BNN algorithm. Section III lists significant works in literature and section IV describes the methodology and the strategies adopted in this project. In Section V we show our system analysis and the achieved results. Finally, Section VI is dedicated to underline the conclusions and the future directions of this work.

II. Background

A. FINN and BNNsplit

In our study presented in BNNsplit [5], we have obtained interesting reductions in terms of occupied area and latency, as well as dissipated dynamic power. The promising results are due to the high level of parallelism and the reduced latency of FPGAs, which made them the best solution to this problem. This led us to the division of the neural network we chose among those proposed in FINN [6] (shown in Fig. 1), opening the possibility of transferring the architecture to a distributed system. The use of more PYNQ boards allowed us to reach a moderate reduction of the dynamic power dissipated, and adds a further level of parallelism to the single board solution. These results are promising in view of a potential use of the system to accelerate neural networks that require more resources than those offered by a single FPGA.

The chosen strategy for the division of the convolutional neural network was to split it between two layers, and more specifically we looked for the point in the neural network where there is the smallest amount of data transferred between the two layers. This led us to chose our split point between the last convolutional layer and the first fully connected layer,

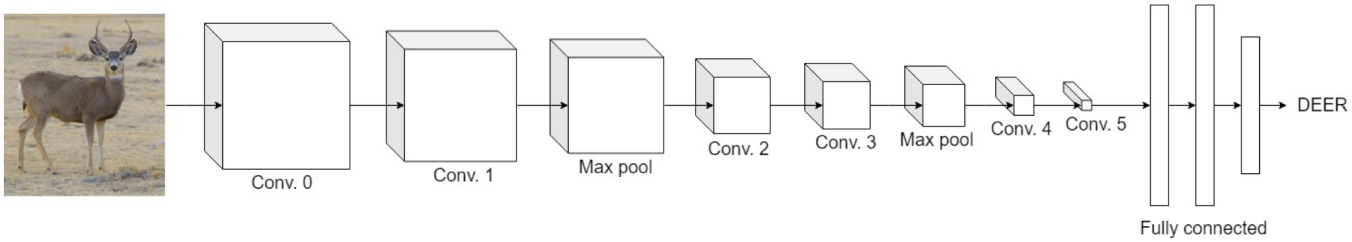

Fig. 1. The topology of the CnvW2A2 BNN: 6 convolutional layers followed by 3 fully connected layers.

since the computations performed in the fully connected part of the neural network do not affect the size of the data - while the convolutions are used to reduce the size of the original data and extract the features.

B. FARD

Fog Acceleration through Reconfigurable Devices (FARD) [4] is a fog computing distributed system designed to allow seamless cooperation across heterogeneous fog computing nodes. Inside FARD, two different aspects are coexisting: hardware acceleration and distributed run-time management. The system is built upon the concepts of *peer*, *task*, *event* and *application*. An application is a collection of tasks, where tasks can be different and can be replicated across the distributed system. All the tasks can communicate with each other through a dedicated overlay network sending messages and generating events. Each task is a python executable that can leverage multiple python files and that can load bitstreams onto the FPGA. Within this context, a *peer* is an instance of a task that is running inside the FARD distributed system.

III. Related Works

Using the same nomenclature as in the survey [7] that lists the most recent approaches to map CNNs on FPGAs, our solution supports BNN models targeting Xilinx SoC devices with a Streaming architecture. With respect to similar solutions listed in the survey, we recognized the following limitations: fpgaConvNet and DeepBurning does not support multi-FPGA systems and executing different parts of a CNN with the multi-bitstream design requires complete reconfiguration of the FPGA; Haddoc2 unrolls its input and output feature maps and the dot products of convolutions completely, increasing the DSPs and the on-chip storage required to map the CNN, limiting the size of models that can be mapped; AutoCodeGen supports data-driven control mechanisms for each CNN stage, but it uses single-layer parametrized blocks and does not support multi-bitstream designs and distributed systems.

Although the variety of works of CNNs on FPGA is very higher, only a few papers exploit a system with multiple FPGAs. This is the case of [8], [9] where a deeply pipelined multi-FPGA architecture is used both for training and inference of CNNs. However, deeply pipelined multi-FPGA architecture fits only a specific class of algorithms within the distributed scenarios and authors described a custom communication infrastructure to deal with distributed nodes communication, instead of trying to generalize the technique.

Finally, other works, such as [10], [11], extended FINN in the past. The former proposes an extension to the original version of the framework with support for arbitrary precision and more flexibility in the end architecture and target platforms, including hardware cost estimation for given devices. The latter proposes an extension for parametric hardware architectures of Long Short-Term Memory layers on FPGAs which offers full precision flexibility and allows for performance scaling offering different levels of parallelism within the architecture.

IV. Methodology

A. FARD improvements

In order to make FARD more suited for our purpose we extended it to add support for *jobs* and *pipelines*. A *job* has the same role as a task, the key difference lies in its life cycle. A *job* has the main purpose of processing a fixed-size batch of data, after the processing completes, the *job* is killed. As depicted in Figure 2, a *pipeline* is a set of *jobs*, each one with a specific position in the flow of data.

These extensions are needed in order to share the system among users. With *tasks* the user would need to manually kill the *task* in order to free the system resources. With *jobs* the system resources are freed up as soon as the data have been processed. More precisely the life cycle of a *job* is the following: the FARD's API receives a job request in which is specified the processing load (number of chunks of data to process, in our case a set of images to classify) and the actual data to process. After the message as been received by the node manager (the component responsible for orchestrating *tasks* and *jobs*) the *jobs* are spawned and the *pipeline* is orchestrated (each *job* receives is configuration in the *pipeline* configuring itself and uploading the bitstream to the FPGA). Once every *job* has received the configuration, the first *job* (position 0 in the *pipeline*) starts processing the data in its input queue (filled by the node manager with the data sent to the API) and starts to fill its output queue, the second *job* does the same until the last *job* which output queue is automatically sent to the API's caller, ending the *pipeline* and its *jobs*.

From the viewpoint of the end user the API has not been extended in a drastic way, a new method is now available: `api.send_job_data(app_id, job_data)`

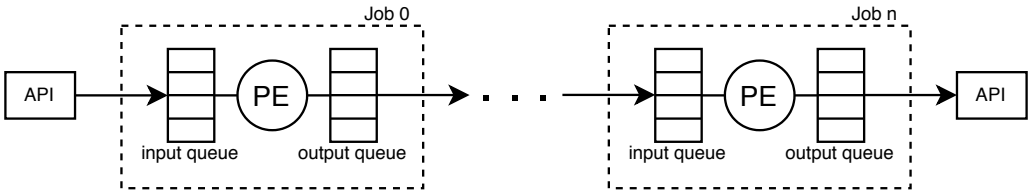

Fig. 2. FARD extension with pipelines. A *pipeline* is a set of *n jobs*, each one with a specific position in the flow of data.

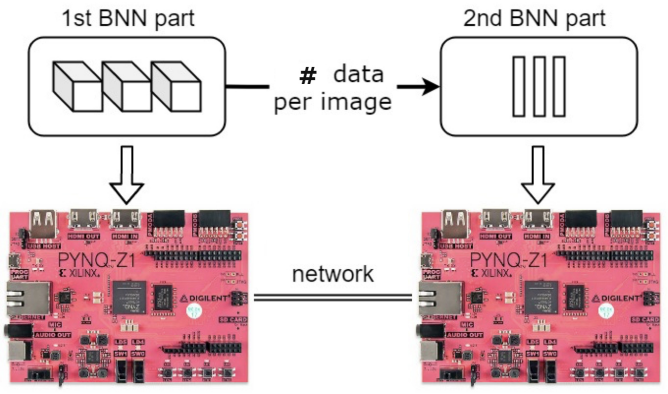

Fig. 3. System design: division of the CnvW2A2 BNN in two parts implemented on two connected PYNQ-Z1 boards.

which is used to send the input data to the *node manager*. Apart from that the only difference is the `FardJob` class which extends `FardTask` and provides the method `job._parse_config(config)` in which the user provides the implementation of the job's function (the position in the pipeline can be accessed from the property `self.config.pipeline.position`).

B. Proposed design

Figure 3 shows the system design used to collect the results described in the next section. The selected BNN was divided into two parts and 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. The stream is then processed enabling task-level pipelining with the `#pragma HLS DATAFLOW`, allowing the overlapping of the redundant BNN operations on the multiple feature maps in order to increase concurrency and throughput. The first part of the BNN takes one or more pictures as input, which is quantized and packed into a 32x32 image on the host side. The image(s), now represented by an `ap_uint<64>` vector, are then converted on the kernel into a `stream<ap_uint<64>>`, which will go through the operations performed by the layers of the first kernel. Once the processing is completed, the last layer returns for each image a stream of data that represents the output of the first kernel. The resulting stream must be transformed again to a `stream<ap_uint<64>>`, in order to convert it back to an `ap_uint<64>` memory block and transfer it to the following board throughout the network. Both the conversion to `stream<ap_uint<64>>` and the following one to `ap_uint<64>` must take into account the number of channels for the output feature maps, the channel size and the size of the output feature maps. These values are used to calculate the number of bits of the partial result, in order to compute both the conversions correctly without losing relevant information.

V. Evaluation

The neural network we chose is the same used in [5] (shown in Fig. 1), and it was divided as mentioned in previous sections. Fig. 4a shows the increase of resource utilization in the first part of the network, while moving the split point at the end of different layers. The chosen split point is located at layer 5 - the increase of resource utilization generated by the fully connected layers (6 to 8) was not relevant and hence not reported in the figure. The significant growth of the area occupied on the board by the application is balanced out by the massive reduction in size of the data transferred between the layers, that begins with more than 12000 Bytes/img in output at layer 0 and reaches a value of only 64 Bytes/img at layer 5 [5]. The percentage values of resource utilization are shown in detail in Table I. While LUT and FF experience an almost linear growth, we can notice that the BRAM grows almost exponentially: this is due to the increase of parameters loaded on the board by adding more layers. A close-up on the difference in terms of resource usage on a single board between the full network and both the convolutional and fully connected portions of the net is presented in Fig. 4b. Both the fully connected and the convolutional parts leave some free space on the board - more significant for the fully connected, that could be eventually occupied by different applications by means of partial reconfiguration.

For what concern the modified FARD framework, a new application (`two_stage_bnn_app`) has been created and configured as followed:

- node `0` has been configured to be the first node in the pipeline and to start the job `first_stage` with the role of computing the first stage of the classification (first half of the neural network)
- node `1` has been configured to be the second (and last) node in the pipeline and to start the job `second_stage` with the role of computing the second (last) stage of the classification (second half of the neural network)

The average response time on 30 runs of 20 images each is 0.084s with a standard deviation of 0.0075s. As a further

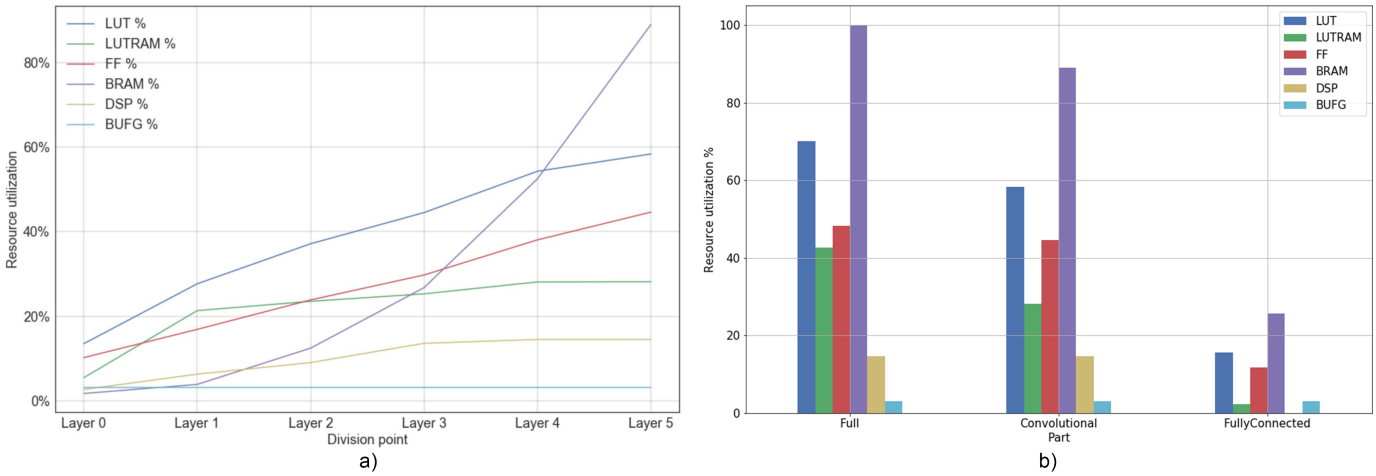

Fig. 4. **a)** On left, the increase of resource utilization at different split points. The BRAM grows almost exponentially with the rise of the amount of parameters loaded on the board. **b)** On right, the resource utilization of full BNN and the two splitted parts when the division is performed after Layer 5. The Convolutional part consumes nearly 10% less resources than the full network, while the fully connected part has a much lower resource usage.

TABLE I
RESOURCE UTILIZATION OF THE 1^{st} FPGA AT DIFFERENT SPLIT POINTS.

Split	LUT%	LUTRAM%	FF%	BRAM%	DSP%
Layer 0	13.55	5.51	10.26	1.79	2.73
Layer 1	27.69	21.36	16.93	3.93	6.36
Layer 2	37.15	23.56	16.93	12.50	9.09
Layer 3	44.51	25.32	29.77	26.79	13.64
Layer 4	54.27	28.13	38.07	52.50	14.55
Layer 5	58.35	28.17	44.62	88.93	14.55

improvement of these performances, it is possible to delegate the message-based communication to the sole purpose of orchestrating the system behavior and dedicate a socket-based communication for the data chunks transfers.

VI. CONCLUSION

In this work, we presented an evaluation of the resource usage when a CNN, more specifically a BNN, has to be split among different nodes of an FPGA-based distributed system. In particular, the resource utilization progressively increases when the split point moves forward among the convolutional layers. There is a significant growth in BRAM utilization, due to the necessity to load weights and activations for the new layers. Moreover, we described how to extend the FARD framework to support job-based applications, like those related to CNN-based algorithms. Although the communication latency results are not optimal, we have laid the foundations for building an optimized version that can benefit from future use of sockets-based communication. In addition to this, we plan a more in-depth analysis of larger CNN architectures, where the mentioned improvements together with a splitting strategy able to balance hardware resources, communication load, and system throughput could bring to even better results.

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] 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

[3] M. Kim and P. Smaragdis, "Bitwise neural networks," *CoRR*, vol. abs/1601.06071, 2016. [Online]. Available: http://arxiv.org/abs/1601.06071

[4] S. Barbieri, F. Casasopra, R. Brondolin, and M. D. Santambrogio, "Fog acceleration through reconfigurable devices," in *2019 IEEE 5th International forum on Research and Technology for Society and Industry (RTSI)*. IEEE, 2019, pp. 138–143.

[5] G. Fiscaletti, M. Speziali, L. Stornaiuolo, M. Santambrogio, and D. Sciuto, "Bnnsplit: Binarized neural networks for embedded distributed fpga-based computing systems," in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2020.

[6] 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.

[7] S. I. Venieris, A. Kouris, and C.-S. Bouganis, "Toolflows for mapping convolutional neural networks on fpgas: A survey and future directions," *arXiv preprint arXiv:1803.05900*, 2018.

[8] C. Zhang, D. Wu, J. Sun, G. Sun, G. Luo, and J. Cong, "Energy-efficient cnn implementation on a deeply pipelined fpga cluster," in *Proceedings of the 2016 International Symposium on Low Power Electronics and Design*. ACM, 2016, pp. 326–331.

[9] T. Geng, T. Wang, A. Sanaullah, C. Yang, R. Xu, R. Patel, and M. Herbordt, "Fpdeep: Acceleration and load balancing of cnn training on fpga clusters," in *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2018, pp. 81–84.

[10] F. Kästner, B. Janßen, F. Kautz, M. Hübner, and G. Corradi, "Hardware/-software codesign for convolutional neural networks exploiting dynamic partial reconfiguration on pynq," in *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 2018, pp. 154–161.

[11] V. Rybalkin, A. Pappalardo, M. M. Ghaffar, G. Gambardella, N. Wehn, and M. Blott, "Finn-l: Library extensions and design trade-off analysis for variable precision lstm networks on fpgas," in *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2018, pp. 89–897.