Partial Reconfiguration for Energy-Efficient Inference on FPGA: A Case Study with ResNet-18

Zhuoer Li, Sébastien Bilavarn

Partial Reconfiguration for Energy-Efficient Inference on FPGA: A Case Study with ResNet-18

ZHUOER LI[1] AND SÉBASTIEN BILAVARN[1]

[1]LEAT, Université Côte d'Azur, Sophia Antipolis, France

Efficient acceleration of deep convolutional neural networks is currently a major focus in Edge Computing research. This paper presents a realistic case study on ResNet-18, exploring Partial Reconfiguration (PR) as an alternative to the standard static reconfigurable approach. The PR strategy is based on sequencing the layers of the DNN on a single reconfigurable region to significantly reduce the amount of Programmable Logic (PL) resources required. Results demonstrate that PR-based acceleration can reduce FPGA resource usage by over 6 times, power consumption by 3.2 times, and the corresponding global energy cost by 2.7 times, with only a 17.5% increase in execution time. This approach shows great potential for further reductions in area and power consumption.

1. INTRODUCTION

With the increasing prevalence and popularity of edge Artificial Intelligence (AI) applications, there is a clear move towards enabling deep learning models on edge devices. In this context, Convolutional Neural Networks (CNNs), at the origin of a lot of applications ranging from image recognition to real-time video analysis, have started to be widely used. Training deep CNNs (DNNs) used to come with inherent challenges such as the vanishing gradient problem. Residual Networks (ResNets), originally introduced by Microsoft Research, have emerged as a possible solution and won the ImageNet competition in 2015 [1]. They introduce the concept of residual block (Fig. 1) to address the gradient problem arising when stacking a large number of layers. The basic principle is based on adding the residual of a function F(x) = H(x) - x in the learning process. Residual blocks are therefore present in the original topology and stacked together to form a full ResNet model (Fig. 3).

ResNet enables deep learning models with tens or hundreds of layers to be easily trained and reach better accuracy. However, the inherent size and complexity behind this create challenges for their actual implementation, especially to allow practical embedded applications for mobile devices. Works quickly started to address hardware acceleration and especially implementation on Field-Programmable Gate Arrays (FPGAs), to greatly improve energy and processing efficiency. FPGAs offer a unique combination of high performance, low power consumption, and adaptability, making them well-suited for deploying complex CNNs like ResNets on edge devices. The consideration of Partial Reconfiguration (PR) further extends the usual flexibility and efficiency of FPGAs, allowing for the dynamic modification of configuration bitstreams at runtime. This feature can be extremely beneficial for edge applications, enabling the optimization of processing capabilities without interrupting ongoing operations, thus significantly improving area and efficiency.

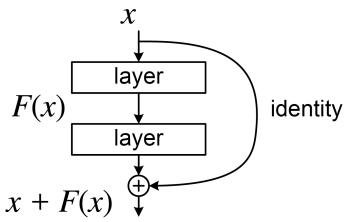

Fig. 1. Residual Block in a deep Residual Network, with skip connections that perform identity mappings merged with the layer outputs by addition.

However, deploying ResNets on FPGAs for edge applications, particularly when considering Partial Reconfiguration (PR), presents unique challenges that demand advanced methodologies for efficient system-level exploration and mapping. Proper utilization of PR has the potential to yield significant benefits beyond those achievable with traditional static FPGA allocation. However, PR also brings a very high level of a design challenge to identify relevant mappings in a much larger design space, which escalates rapidly with the complexity of the neural network and the number of hardware functions involved.

In this work, we address this with the help of a previously defined methodology for the efficient mapping of large and complex application graphs on manycore reconfigurable accelerated systems supporting PR [15]. The distinctive contribution of this paper is to address an evaluation of such PR methodology on ResNet-18 and assess the corresponding impact of PR on processing and energy efficiency. The paper is organized as follows. Section 2 first reviews various state-of-the-art contributions on ResNet implementation on FPGA pointing out the issue of area and size of programmable logic. Section 3 addresses the global hardware software investigation methodology including the central Design Space Exploration (DSE) approach dealing with PR. Section 4 reports experiment studies of ResNet-18 with a comparison of PR and static solutions. Section 5 finally draws the main conclusions of the results with future direction for research.

Table 1. ResNet implementation on FPGA platforms

Work	Platform	Frequency	Network	T_{1frame} (ms)
[2] (2017)	FPGA Intel Arria 10 GX115	200 MHz	ResNet-9	3.17
[3] (2018)	FPGA Xilinx PYNQ	100 MHz	ResNet-18 (Converted, 8 bits)	-
[4] (2018)	FPGA Intel Stratix V	-	ResNet-50	31.82
	FPGA Intel Arria 10	-		12.7
[5] (2019)	FPGA Intel Arria 10	-	ResNet-20	0.22
[6] (2020)	FPGA Xilinx ZC706	166 MHz	ResNet-32	0.60
			ResNet-152	-
[7] (2022)	FPGA Xilinx XC7VX980T	100 MHz	ResNet-101	-

2. RELATED WORKS

[2] (2017) is one of the first approaches describing inference acceleration on FPGA with a solution to accelerate any construction of ResNet based on using OpenCL. Implementation of a relatively small ResNet-4 topology (9 layers) for the CIFAR-10 dataset (32x32 images) on Intel Arria 10 GX115 device reports 315.3 fps. [3] (2018) proposed an approach based on the Apache TVM compiler for Deep Learning and the Versatile Tensor Accelerator (VTA), a generic, modular, and customizable architecture for TPU-like accelerators. 8-bit ResNet-18 inference implemented on a low-cost Xilinx Zynq FPGA (PYNQ board, 100MHz) achieves 51 GOPS. [4] (2018) is another contribution addressing more specifically convolution-level loop optimizations to improve the global efficiency and performance of CNN accelerators. Their mapping study of ResNet-50 reports 31.82 ms/image (31.4 fps, 243.3 GOPS) on Intel Stratix V GXA7, and 12.7 ms/image (78.7 fps, 611.4 GOPS) on Intel Arria 10 GX 1150.

[5] (2019) is another study investigating several deep learning FPGA deployment inference examples, more especially in the field of autonomous driving with traffic sign classification and detection. Implementation of ResNet-20 (32x32) and ResNet-32 (64x64) reports respectively 4637.72 and 1673.56 fps on an Intel Arria 10 FPGA platform. To overcome the computational complexity and implementation cost on FPGA, the authors in [6] (2020) use fast 2-D Winograd and FFT algorithms to reduce the number of multiplications in the convolutional layers. With this approach, combined with a specific accelerator architecture and design space exploration stage to achieve an optimal level of parallelism, a ResNet-152 model can reach 130.4 GOPS on the Xilinx ZC706 platform at 166 MHz. In more recent works [7] (2022), a multi-Computing Engine architecture based on an array of Processing Elements is proposed along with its associated mapping methodology, in a way to improve also on the convolution performance. Based on this design, a ResNet-101 accelerator is implemented achieving 600 GOPS on a Xilinx XC7VX980T device at 100 MHZ.

It is notable from these works that convolution layers are a major concern, but also that size is a very typical feature of ResNets which implies most of the time the use of rather large FPGA devices. Some works have therefore further investigated the use of multi-FPGA platforms. [9] (2018) for example is an early approach dealing with processing and memory capacity using highly quantized neural networks and supporting implementation on multiple FPGAs. Deployment of ResNet-18 on a platform using three Intel Stratix V 5SGSD8 reports 16.1 ms per frame (65.2 fps). [8] (2021) is another approach targeting a multi-FPGA system made of small low-cost ZYNQ boards (PYNQ cluster). Implementation of a ResNet-50 inference accelerator is achieved by dividing layers into multiple boards so that the execution can be efficiently pipelined. With this deployment, stream processing on ResNet-50 using four PYNQ boards achieves 292 GOPS performance and 75.1 fps throughput.

As it can be seen in [9], advanced quantization is also being widely used to reduce on-chip storage and improve computation throughput. Indeed, in many cases, a low-precision representation (1-2 bits per parameter) of weights and other parameters can achieve similar accuracy while requiring fewer resources. Since FPGAs provide efficient support for bitwise operations and can handle arbitrary-precision representations of numbers, using quantized values with lower precision can ease the pressure on resources. An application study is described for example in [10] where quantization techniques that use lower than 8-bit precision are investigated. Evaluation of ResNet-18 and ResNet-50 reports respectively 27.8 fps, 13.3 fps on PYNQ-Z2 and 214.8 fps, 109.1 fps on ZCU102 Xilinx platforms.

Binary Neural Networks (BNN) have also been investigated in order to save storage and computation resources on FPGA with Xilinx FINN HLS library [11] (2018). BNNs extend the concept to extremely low bit quantization (binary weights), leading to very compact implementations. A mapping study of ResNet-50 is reported for example in [12] achieving up to 50 fps on an Alveo U280 device.

Compression is another effective technique for resource optimization of DNNs. In [13] for example (2019), the authors introduce an approach to significantly reduce the size of parameters and memory footprint of the models, based on compressing the DNN parameters down to two bits with little accuracy drop. Implementation study of ResNet-18 reports 20.48 fps on a ZedBoard platform.

Finally, a last attractive option is to exploit Partial Reconfiguration (PR). The ability of PR to reduce area has been widely investigated in previous research with successful results in many

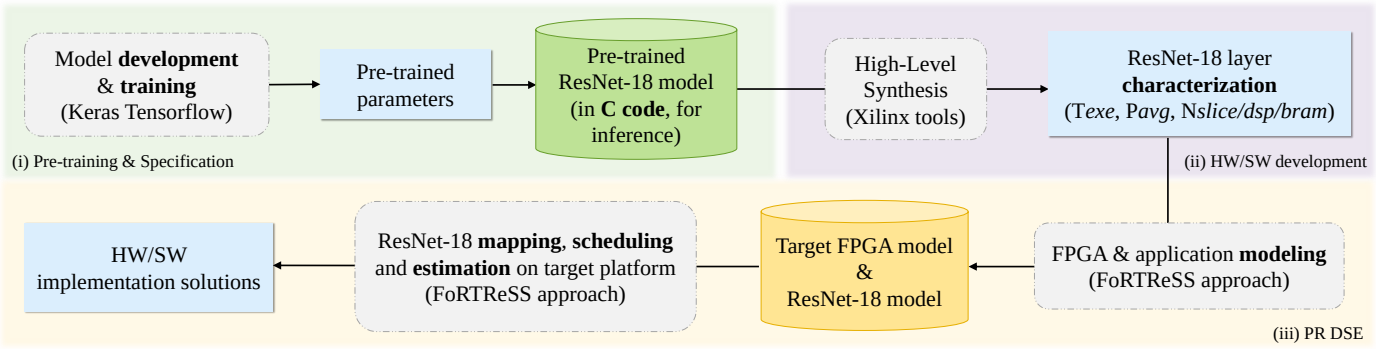

Fig. 2. Overview of the global PR workflow. First, training the ResNet-18 model using Keras produces weights and bias values. Then, full ResNet-18 C code is derived and used to generate RTL IPs with HLS, which subsequently allows to derive inference time, FPGA logic resources and power. This characterization process is then used for DSE with a specific PR methodology (FoRTReSS).

application domains. However, PR has hardly been applied to the field of ANNs despite the evident potential to drastically save hardware area, especially for large DNNs. The only existing contribution in the field of large Neural Networks, up to our knowledge, is given by fpgaConvNet [14] (2019) which is an approach originally developed to support automated design and mapping of CNNs on FPGA. Here, due to limited FPGA's computational and memory resources, straightforward reconfiguration of the whole FPGA at run time is additionally used to make large DNNs practically fit in actual existing devices. Application on a ResNet-152 topology achieves 156,40 ms/image (188.18 GOPS) on Zynq 7045 (125 MHz).

Table 1 summarizes the different contributions of ResNet FPGA acceleration based on techniques such as advanced quantization and multi-FPGA systems discussed above, emphasizing their respective contributions in terms of implementation and performance. Aside from inference time and accuracy, size and resource requirements are also major constraints for energy-efficient mapping neural networks, especially on FPGA. Especially for deep Neural Networks such as ResNets, large devices or multi-FPGA approaches combined with existing size reduction techniques are unavoidable to let their practical implementation on FPGAs and more generally on embedded systems. Despite the overall effectiveness of existing techniques, an even greater potential resides in the relevant exploitation of PR. By sequentially reconfiguring layers on a rightly partitioned FPGA device, the potential for size reduction is theoretically proportional to the number of layers. We address therefore in the following the use of such an advanced FPGA mapping methodology capable of exploiting PR, and its application to a large DNN topology (ResNet-18).

3. PARTIAL RECONFIGURATION FLOW

To identify easily a solution in the large and complex PR design space, we use High-Level Synthesis (HLS) in addition to a methodic Design Space Exploration (DSE) approach for the system-level exploration of heterogeneous architectures with reconfigurable hardware acceleration and PR defined in previous work [19]. The entire ResNet-18 specification is developed in pure synthesizable C code derived from the iSoC open-source project [20] and the underlying global PR workflow is illustrated in Fig. 2. Development and modeling of ResNet-18 with this framework is thus presented in detail in the following.

A. Development of a quantized model

Deployment of ResNet-18 involves first training the model using Keras. C code is thus manually developed to allow fast RTL synthesis with HLS. The original C model is further improved on quantization to reduce the global size of the model, allow faster inference, less power, and target embedded devices without the need of any hardware floating-point operators. The development of the C reference code focuses primarily on the convolutional and Batch Normalization (BN) layers, but also includes fully-connected, global average pooling and addition layers. The entire ResNet-18 topology is illustrated in Fig. 3. Operations such as addition, subtraction, multiplication, division, and square root are all processed using 16-bit fixed-point arithmetic (Q6.9). Weights and bias values of convolutional layers are stored using 8-bit fixed-point values (Q2.5). Weights for the BN layers (gamma, beta, mean and variance) use a 16-bit fixed-point format. This aims at reducing memory usage while keeping the same inference accuracy. Less than 1% deviation is reported compared to the original float-point specification (86.3% for the floating-point vs 85.9% for the fixed-point).

The notion of ResNet is based on basic residual blocks. There are primarily two types of residual blocks in this topology: convolutional residual blocks (green in Fig. 3) and identity residual blocks (orange in Fig. 3). The identity residual blocks consist of two convolutional layers with 3x3 kernels, associated with a skip connection (adding the input and output of these two convolutional layers). Convolutional residual blocks are composed of two convolutional layers with 3x3 kernels, adding a skip connection that includes a 1x1 convolution. This is necessary because the number of input and output channels of the convolutional residual blocks differ. Hence a 1x1 convolutional layer is required to equalize the channel dimensions without altering the size of feature maps, thus enabling the addition of skip connections. For HLS function decomposition, we partition the network topology in such a way that each convolution layer and its subsequent BN layer define a single block, and synthesize each block as distinct hardware functions. This block-based approach allows therefore the creation of a wide range of ResNet network topologies with various numbers of layers.

In terms of accelerator design and HLS, we explore loop-level parallelism which remains a manual process. 2D convolutional layers are the most computationally demanding tasks for the global accelerator architecture. As shown in Algorithm 1, the convolutional layer is based on six nested loops that process output channels, output feature maps (rows and columns), in-

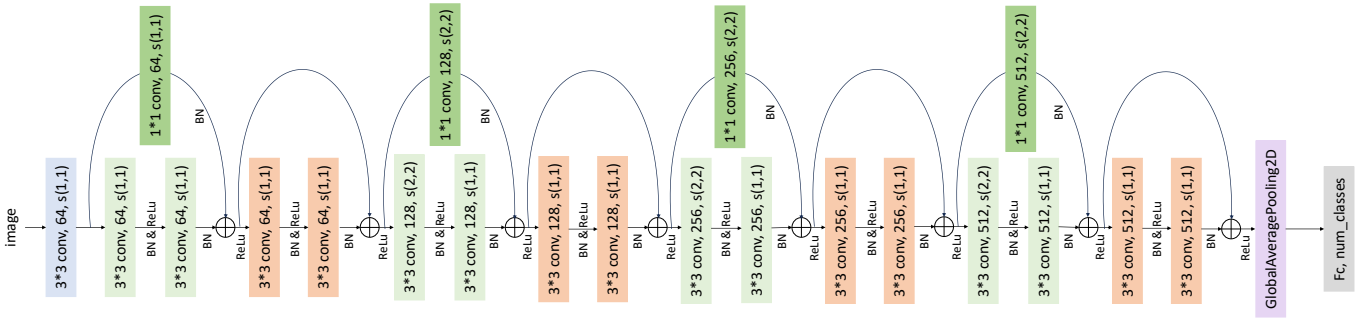

Fig. 3. ResNet-18 topology

Algorithm 1. 2D convolution pseudo-code

```
for k ← 0 to output_channel_size do
    for pos_y ← 0 to output_height_size do
        for pos_x ← 0 to output_width_size do
#pragma HLS pipeline
            for z ← 0 to input_channel_size do
                if z == 0 then
                    kernel_mac ← 0
                for y ← 0 to kernel_size do
                    for x ← 0 to kernel_size do
                        kernel_mac = kernel_mac + input(z, pos_y, pos_x) * kernel(k, z, y, x)
```

put channels, and K*K convolution kernels. Pipeline directives are primarily applied at the input channel level to achieve an optimal trade-off between performance and the utilization of on-chip resources. However, the specific placement of these pipeline directives can vary depending on the memory demands and the computational complexity of different convolutional layers. Similarly, pipeline parallelism is also considered for the BN layers, as shown in the pseudocode of Algorithm 2.

Algorithm 2. Batch Normalization pseudo-code

```
for c ← 0 to output_channel_size do
    for h ← 0 to output_height_size do
        for w ← 0 to output_width_size do
#pragma HLS pipeline
            r_sqrt ← fp_sqrt(variance(c) + ε)
            r_sub ← fp_sub(input(c, h, w), mean(c))
            r_norm ← fp_div(sub, sqrt)
            r_mul ← fp_mul(gamma(c), norm)
            result ← fp_add(tmp, beta(c))
```

B. Design Space Exploration of PR mappings

Identifying optimal deployment solutions is based on an existing methodology called FoRTReSS [15] [16] [17]. This DSE framework allows the analysis of different application deployments with fully implementable real-time and energy-efficient solutions. It automatically defines an extensive set of possible Reconfigurable Regions (RR) from the RTL descriptions of hardware tasks (resulting from HLS). It also generates mappings on different cores and RRs, enabling relevant assessments of performance, area, power, and energy for both software and hardware functions, as well as the full system implementation.

Full hardware task implementation, modeling, and execution on FPGA rely on HLS and lower-level FPGA synthesis tools (Vivado, Vivado HLS, SDSoC). The set of hardware and software functions can then be realistically characterized by synthesis reports and/or execution on real FPGA and CPU cores. The following section provides a detailed breakdown of this characterization process for the ResNet-18 specification.

C. Modeling ResNet-18 mapping on ZCU102

A collection of eight residual blocks (four convolutional residual blocks and four identity residual blocks) with a total of 18 layers constitutes a full ResNet-18 topology (Fig. 3). As a large combination of layers that are processed one after the other, a seemingly attractive option for PR is to schedule each layer execution only using a few RRs to greatly reduce the total resource occupation.

It should be noted here that ResNet needs tremendous FPGA resources which significantly restricts the neural network from fitting in existing FPGAs. This is the case for the considered ResNet-18 topology exceeding CLB and BRAM capacity of ZCU102 respectively of 307% and 282%. In the following, we therefore consider a serial reconfiguration of each layer on a smaller fraction of FPGA logic instead of implementing the entire topology statically. Each layer (convolution, pooling, fully connected) is treated as an individual task that can be potentially mapped on a single RR.

The target platform is a ZCU102 device from the Zynq Ultrascale+ family (quad-core Arm Cortex-A53, XCZU9EG-FFVB1156-1). The corresponding platform model is composed of the Programmable System (PS), the Programmable Logic (PL), and the reconfigurable controller. Details of PS and PL model compositions are provided in [17] and [16]. Power characterization is derived from logic synthesis tools. As PR efficiency is strongly dependent on reconfiguration speed, we consider a very fast optimized reconfiguration controller (UPaRC, Ultra-fast Power-aware Reconfiguration Controller [18]) in place of the original Xilinx PCAP controller.

Table 2 reports all values of latency, power, and FPGA resources for each convolutional block of Fig. 3, considering a ZCU102 device for mapping. For this ResNet-18 model, used as input for DSE, values of execution times and the corresponding FPGA resources come from direct HLS. The C code of each convolution block is used to generate the corresponding RTL

Table 2. Characterization of hardware tasks for ResNet-18 on ZCU102 (1 RR: 8880 CLBs, 648 BRAMs, 432 DSPs)

Task (i)	Execution unit (j)	$T_{i,j}$ (ms)	$P^{idle}_{i,j}$ / $P^{run}_{i,j}$ (mW)	N^{clb} / N^{bram} / N^{dsp}
BlockConv0[1](3x3, 64)	RR1	3.935	51 / 215	1874 / 60 / 28
BlockConv1_SC (1x1, 64)	RR1	1.314	51 / 294	2194 / 60 / 65
BlockConv1_2 (3x3, 64)	RR1	3.937	55 / 960	8639 / 64 / 410
BlockConv1_3 (3x3, 64)	RR1	3.937	55 / 972	8485 / 64 / 425
BlockIdenti1_1 (3x3, 64)	RR1	3.937	55 / 973	8792 / 60 / 402
BlockIdenti1_2 (3x3, 64)	RR1	3.937	55 / 947	8322 / 60 / 387
BlockConv2_SC (1x1, 128)	RR1	0.658	51 / 294	2348 / 65 / 34
BlockConv2_2 (3x3, 128)	RR1	1.972	53 / 745	8775 / 34 / 117
BlockConv2_3 (3x3, 128)	RR1	64.643	52 / 288	4075 / 238 / 19
BlockIdenti2_1 (3x3, 128)	RR1	64.479	52 / 408	4498 / 263 / 20
BlockIdenti2_2 (3x3, 128)	RR1	64.643	52 / 344	3963 / 259 / 19
BlockConv3_SC (1x1, 256)	RR1	0.331	53 / 518	3217 / 132 / 129
BlockConv3_2 (3x3, 256)	RR1	12.381	52 / 444	3447 / 213 / 19
BlockConv3_3 (3x3, 256)	RR1	63.829	53 / 430	6446 / 390 / 19
BlockIdenti3_1 (3x3, 256)	RR1	43.677	51 / 628	4546 / 421 / 19
BlockIdenti3_2 (3x3, 256)	RR1	43.677	51 / 547	4209 / 421 / 19
BlockConv4_SC (1x1, 512)	RR1	3.707	53 / 641	3497 / 92 / 257
BlockConv4_2 (3x3, 512)	RR1	42.617	52 / 371	4197 / 256 / 10
BlockConv4_3 (3x3, 512)	RR1	63.430	53 / 340	3628 / 632 / 19
BlockIdenti4_1 (3x3, 512)	RR1	63.429	53 / 351	3400 / 516 / 19
BlockIdenti4_2 (3x3, 512)	RR1	63.430	53 / 325	3242 / 488 / 19

[1] Each block contains a convolutional layer followed by a batch normalization layer.

hardware task and the associated power values (P^{idle}, P^{run}) are derived from Vivado post-synthesis power estimations. From this characterization, DSE defines an exhaustive set of candidate RRs that can host the different hardware tasks. Then full scheduling and mapping of potential PR solutions are determined for deployments ranging from 0 to multiple RRs (0 RR meaning full software execution). As layers are processed sequentially one after another, it can be noted here that only the PR solution based on one RR is relevant and provides the best performance/energy trade-off. Therefore the mapping solution considered in the following is based on this specific RR illustrated in Fig. 4, encompassing 8880 CLBs, 648 BRAMs, and 432 DSPs (around 30% of the original ZCU102 logic resources).

4. VALIDATION STUDY

A. Power, performance, area

The previous ResNet model is fed into the PR-based exploration tool and the generated different hardware/software solutions are compared in the following. Table 4 reports the global characteristics of *PR* (dynamically reconfigurable accelerator, 100MHz), *Static* (statically reconfigurable accelerator, 100MHz), and *Software* execution (using one Arm Cortex-A53 CPU core, 1200 MHz) in terms of performance, power, and energy. Table 3 details the resource occupation of the two types of hardware solutions, PR and static. For the PR solution, the solution with 1 RR is chosen for comparison, as among the solutions with 0 to 5 RRs, it demonstrates the minimal area and the lowest energy consumption.

First, compared to the software implementation, PR is 20 times faster (731 vs. 14742 ms) but consumes 3.3 times more power (728 vs. 220 mW). Consequently, PR solution is 6.1 times more energy efficient than software (532 vs. 3246 mJ).

In comparison to static acceleration, PR is 17.5% slower (731 vs. 622 ms). This difference represents the total PR reconfiguration overhead, as the additional latencies are entirely due to the consecutive reconfigurations of the single RR used. The parallelism of all hardware tasks is identical for both static and PR solutions. In terms of programmable logic resources, PR requires 12 times fewer CLBs (8880 vs. 105300), 8 times fewer BRAMs (648 vs. 5136), and 6.2 times fewer DSP blocks (432 vs.

Table 3. FPGA resource occupation of static and PR hardware implementations for ResNet-18 on ZCU102

Implementation	N^{CLB}	N^{BRAM_18k}	N^{DSP}
Static HW	105300 (307%) / 34260	5136 (282%) / 1824	2664 (106%) /2520
PR HW (1 RR)	**8880 (26%)** /34260	**648 (35.5%)** / 1824	**432 (17%)** / 2520

Table 4. Efficiency comparison of software, static and PR hardware implementations for ResNet-18 on ZCU102

Implementation	Execution Time (ms)	Average Power (mW)	Energy Consumption (mJ)
SW	14742	220.2	3246.1
Static HW	**622.47**	2337.5	1455.05
PR HW (1 RR)	731.64	728.4	**532.95**

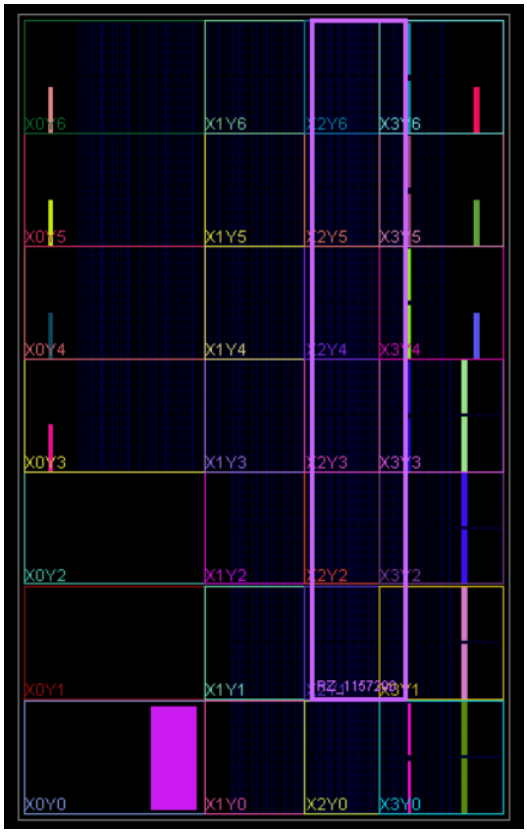

Fig. 4. Layout view of the RR selected for PR implementation of ResNet-18 on ZCU102 (8880 CLBs, 648 BRAMs, 432 DSPs)

2664). As a result, global power consumption is reduced by a factor of 3.2 against static implementation (728 vs. 2337 mW), resulting in a 2.7 times improvement in energy efficiency (533 mJ vs. 1,455 mJ), which corresponds to a 63.4% reduction in energy consumption.

B. PR implementation analysis

Results confirm a real potential for PR improvement on ResNet-18 (63.4% energy savings), and more generally for large ANNs [21]. However, the actual numbers are below expectations, as sharing many network layers with significantly fewer PL resources would ideally result in greater area reduction. Because of the large memory requirements of ResNet-18, and in general of deep neural networks, the RR selected by the exploration tool is still quite large. This limits the potential of FPGA resource savings. At the same time, since the reconfiguration time is proportional to the size of the RR, this also has a negative impact on the overall reconfiguration time.

Therefore, the reconfiguration times associated with this large RR end up representing around 17.5% of the total ResNet-18 inference time. Reconfiguration control plays a critical role and has to be highly optimized to permit extremely fast reconfiguration times. Despite the use of an optimized reconfiguration controller (UPaRC [18]), future works will also address specific scheduling heuristics, such as those described in [22], to further improve reconfiguration latencies.

Finally, it can be concluded that optimal PR efficiency is significantly limited by the very large memory requirements induced mainly by network parameters (weights). An interesting perspective would be to exploit the PR technique differently, by more progressively loading network parameters into local memory on the fly, thereby drastically reducing on-chip storage requirements on the FPGA. This is the main work envisaged in the perspectives.

5. CONCLUSION

This work presents a mapping study of a ResNet-18 DNN model on a ZCU102 FPGA platform with partial reconfiguration. The results indicate notable improvements of PR over static reconfiguration, with more than 6 times reduction in programmable logic resources, 3.2 times reduction in power consumption, and 2.7 times energy savings, accompanied by a 17.5% execution time increase, totally incurred by the reconfiguration overheads. Previous work already introduced the improvement potential of PR with energy savings up to 1.67 times for smaller CNN network benchmarks (MNIST, GTSRB, CIFAR-10) compared to their equivalent static implementations [21]. Additionally, PR also showed increasing energy gains for larger networks, as further confirmed on ResNet-18 in this paper.

These results are therefore encouraging enough to consider PR as a promising technique to significantly improve the energy efficiency of DNNs on FPGA. One essential condition is to achieve fast reconfiguration control and/or scheduling, which can be based on approaches such as [22]. In addition, this study

allows us to envisage even more effective PR usage in perspectives by exploiting reconfiguration differently, specifically at the memory level of DNNs, to drastically reduce this way FPGA size and memory requirements for large embedded neural networks.

REFERENCES

1. K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 2016, pp. 770-778, doi: 10.1109/CVPR.2016.90.
2. X. Li, L. Ding, L. Wang and F. Cao, "FPGA accelerates deep residual learning for image recognition," 2017 IEEE 2nd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC), Chengdu, China, 2017, pp. 837-840, doi: 10.1109/ITNEC.2017.8284852.
3. T Moreau, T Chen and L Ceze, "Leveraging the VTA-TVM Hardware-Software Stack for FPGA Acceleration of 8-bit ResNet-18 Inference", Proceedings of the 1st on Reproducible Quality-Efficient Systems Tournament on Co-designing Pareto-efficient Deep Learning (ReQuEST '18), June 2018.
4. Yufei Ma, Yu Cao, Sarma Vrudhula and Jae-sun Seo, "Optimizing the Convolution Operation to Accelerate Deep Neural Networks on FPGA", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, July 2018.
5. Zhongyi Lin, Matthew Yih, Jeffrey M. Ota, John Douglas Owens and Pınar Muyan-Özçelik, Benchmarking Deep Learning Frameworks and Investigating FPGA Deployment for Traffic Sign Classification and Detection, IEEE Transactions on Intelligent Vehicles, May 2019.
6. L. Lu, Y. Liang, Q. Xiao and S. Yan, "Evaluating Fast Algorithms for Convolutional Neural Networks on FPGAs,", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, April 2020.
7. W. Huang et al., "FPGA-Based High-Throughput CNN Hardware Accelerator With High Computing Resource Utilization Ratio," in IEEE Transactions on Neural Networks and Learning Systems, vol. 33, no. 8, pp. 4069-4083, Aug. 2022, doi: 10.1109/TNNLS.2021.3055814.
8. Y. Fukushima, K. Iizuka and H. Amano, "Parallel Implementation of CNN on Multi-FPGA Cluster," 2021 IEEE 14th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC), Singapore, Singapore, 2021, pp. 77-83, doi: 10.1109/MCSoC51149.2021.00019.
9. Chaim Baskin, Natan Liss, Evgenii Zheltonozhskii, Alex M. Bronstein and Avi Mendelson, "Streaming Architecture for Large-Scale Quantized Neural Networks on an FPGA-Based Dataflow Platform", IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2018.
10. Mengshu Sun, Zhengang Li, Alec Lu, Yanyu Li, Sung-En Chang, Xiaolong Ma, Xue Lin and Zhenman Fang, "FILM-QNN: Efficient FPGA Acceleration of Deep Neural Networks with Intra-Layer, Mixed-Precision Quantization", International Symposium on Field-Programmable Gate Arrays (FPGA '22), 2022.
11. Michaela Blott, Thomas B. Preußer, Nicholas J. Fraser, Giulio Gambardella, Kenneth O'brien, Yaman Umuroglu, Miriam Leeser, Kees Vissers, FINN-R: An End-to-End Deep-Learning Framework for Fast Exploration of Quantized Neural Networks, ACM Transactions on Reconfigurable Technology and Systems, 2018.
12. Lucian Petrica and Tobías Alonso and Mairin Kroes and Nicholas J. Fraser and Sorin Dan Cotofana and Michaela Blott, Memory-Efficient Dataflow Inference for Deep CNNs on FPGA, International Conference on Field-Programmable Technology (ICFPT), 2020.
13. Yao Chen, Kai Zhang, Cheng Gong, Cong Hao, Xiaofan Zhang, Tao Li and Deming Chen, "T-DLA: An Open-source Deep Learning Accelerator for Ternarized DNN Models on Embedded FPGA", IEEE Computer Society Annual Symposium on VLSI (ISVLSI), 2019.
14. Stylianos I. Venieris and Christos-Savvas Bouganis, "fpgaConvNet: Mapping Regular and Irregular Convolutional Neural Networks on FPGAs", IEEE Transactions on Neural Networks and Learning Systems, 2018.
15. F. Duhem, F. Muller, R. Bonamy, S. Bilavarn, "Fortress: a flow for design space exploration of partially reconfigurable systems". Design Automation for Embedded Systems, 301–326, 2015, doi: 10.1007/s10617-015-9160-2.
16. R. Bonamy, S. Bilavarn, D. Chillet, O. Sentieys, "Power Modeling and Exploration of Dynamic and Partially Reconfigurable Systems", Journal of Low Power Electronics, 2016, doi: 10.1166/jolpe.2016.1448.
17. R. Bonamy, S. Bilavarn, F. Muller, F. Duhem, S. Heywood, P. Millet, F. Lemonnier, "Energy efficient mapping on many-core with dynamic and partial reconfiguration: Application to a smart camera", International Journal of Circuit Theory and Applications, Wiley, 2018, doi: 10.1002/cta.2508.
18. R. Bonamy, H. -M. Pham, S. Pillement and D. Chillet, "UPaRC—Ultra-fast power-aware reconfiguration controller," 2012 Design, Automation & Test in Europe Conference & Exhibition (DATE), Dresden, Germany, 2012, pp. 1373-1378, doi: 10.1109/DATE.2012.6176705.
19. FoRTReSS Tool Box, http://fortress-toolbox.unice.fr
20. iSoC - Neuromorphic accelerator design, https://isoc-design.com
21. Zhuoer Li, Sebastien Bilavarn, Improving the Energy Efficiency of CNN Inference on FPGA using Partial Reconfiguration, Design and Architectures for Signal and Image Processing (DASIP 2024), Jan 2024, Munich, Germany.
22. Hariharan, I. and Kannan, M., Algorithms for reducing reconfiguration overheads using prefetch, reuse, and optimal mapping of tasks. Concurrency and Computation: Practice and Experience, Wiley, 2021.