Improving the Energy Efficiency of CNN Inference on FPGA using Partial Reconfiguration

Zhuoer Li, Sébastien Bilavarn

▶ **To cite this version:**

Zhuoer Li, Sébastien Bilavarn. Improving the Energy Efficiency of CNN Inference on FPGA using Partial Reconfiguration. DASIP 2024, Jan 2024, Munich, Germany. hal-04699445

HAL Id: hal-04699445
https://hal.science/hal-04699445v1

Submitted on 16 Sep 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Improving the Energy Efficiency of CNN Inference on FPGA using Partial Reconfiguration

ZHUOER LI[1] AND SÉBASTIEN BILAVARN[1]

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

With the increasing demand for edge AI application scenarios, as the most popular deep learning models, Convolutional Neural Networks (CNNs) need advanced solutions for the deployment of highly energy-efficient implementations. This paper presents a novel approach to improve the efficiency of CNN inference on Field-Programmable Gate Arrays (FPGAs) using Partial Reconfiguration (PR). Our method deconstructs CNN topology into different layers for runtime reconfiguration with fewer resources, aiming to significantly reduce static power and overall energy consumption. To identify the conditions for practical PR efficiency, we present a thorough design space exploration study with three CNN benchmarks, each evaluated across three different implementations. The comparison results demonstrate that our PR approach can achieve up to 3.88 and 1.67 times energy savings compared to software and static hardware implementations, respectively. These results also show that the benefits of PR improve with the depth of the network, suggesting very promising levels of gains as the network gets larger and under the key conditions of using fast optimized reconfiguration controllers and methodical system-level exploration of the increased hardware implementation complexity.

1. INTRODUCTION

In recent years, the popularity of Artificial Intelligence (AI) in various domains has led to a growing demand for research in power efficiency, particularly due to significant computing requirements in edge AI. Embedded deep learning models, especially Convolutional Neural Networks (CNNs), have received a lot of interest from embedded development research which quickly started to investigate reconfigurable accelerators and FPGAs to combine high performance, low energy, and flexibility. In this regard, Partial Reconfiguration (PR) is now a well-identified technique that can be also used to push FPGA adaptivity one step further. This technique allows modifying an FPGA configuration by loading another configuration file, often referred to as a "partial" BIT file, during runtime. In general, the FPGA is partitioned into different areas called Reconfigurable Regions (RR) which are physically independent and could also theoretically be run in parallel. While a RR is running, it can be locally configured independently with a new partial BIT file without compromising the integrity of the other running regions. Therefore, this technique of local dynamic reconfiguration enables the configuration of more system functions on a smaller surface area, consuming less power, and allows running regions in parallel, thereby enhancing performance. Correct exploitation of this means potentially bringing significant additional benefits on top of those resulting from pure static FPGA allocation, but the counterpart is design complexity to identify relevant deployments which can quickly explode with the application complexity and the number of hardware functions.

In this work, we address this with the help of a previously defined methodology for efficient mapping of large and complex application graphs on manycore reconfigurable accelerated systems allowing dynamic and partial reconfiguration [11]. Section 2 first reviews existing works on CNN hardware acceleration in relation to dynamic reconfiguration. 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 three CNN benchmarks and a comparison with other works. Section 5 finally draws the main conclusions of the results with future direction for research.

2. STATE OF THE ART

Despite the very abundant literature investigating the question of FPGA acceleration for Deep Learning, PR only started to be addressed in quite recent works. Relatively few papers have therefore investigated how to benefit from the capabilities of PR to support more specifically the computation cost of CNN processing. Ideally, PR would rely on sequentially scheduling each layer of a CNN onto one or a few RR(s) to reduce the overall size of the reconfigurable area. In other words, the entire share of hardware functions would be mapped dynamically to RRs and no static part would remain in the design, a concept we'll refer to as "reductive PR". However, in practice, mainly due to excessive reconfiguration times, a majority of works restrict the use of PR to a smaller part of the system, referred to as "functional PR".

This is for example the case in [1] (2019) where PR is used to add layers at the convolution level when needed, to allow switching dynamically from shallow to deeper topologies, providing this way an adaptive capacity to adjust the model structure and overall classification accuracy at run-time. Another technique is described in [4] (2021) where a macroblock-based PR implementation is defined to let the dynamic reconfiguration of parts

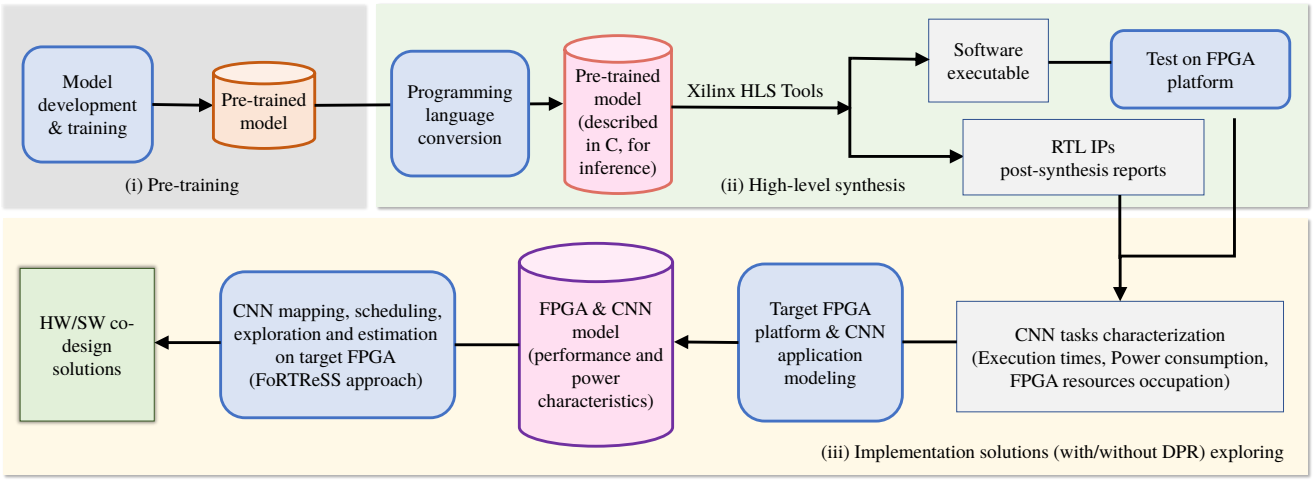

Fig. 1. Overview of our approach workflow. Firstly, we perform training using Keras to process the weights and bias values of the CNN model. Secondly, with the help of HLS tools, we synthesize our CNN model described in C to get the RTL IPs, which subsequently enable us to measure the inference time, power, and resource quantity. These characteristics allow the modeling of target FPGA and CNN layers, which are necessary to explore the implementation solutions using FoRTReSS approach.

of the CNN. Application to a LeNet topology addresses the reconfiguration of two different fully connected layers, enabling a switch between two network versions for either number recognition or letter recognition. Functional PR is also involved in [5] (2014) where a PR strategy is used to implement sigmoid and hyperbolic tangent functions in a Multi-Layer Perception (MLP), or in [2] (2022) and [3] (2020) to exploit different quantization schemes or bit widths (12, 10, 8, 7, 6, 5 bits).

In the category of works addressing a reductive PR approach, fpgaConvNet [6] (2016) is the first proposed approach in the literature, up to our knowledge. In their work, the authors define an automated design flow for mapping CNNs on FPGAs considering design space exploration and High-Level Synthesis (HLS). They employ a formalization technique to automatically explore parallelism such as intra-layer unrolled execution and inter-layer pipeline to tune the performance-resource tradeoff and explore the design space. It should be noted here that, in this approach, the authors intend to map each layer on the entire FPGA without partitioning them into RRs. As a result, they need to employ full FPGA reconfiguration before the execution of each layer. They address the associated reconfiguration latency problem by pipelining execution and reconfiguration to overlap and amortize the associated latency overheads. However, this solution does not *practically* remove the important reconfiguration times of the full FPGA and especially their associated energy cost. For instance, the reconfiguration latency for the entire XC7Z020 device using PCAP on a ZedBoard platform is around 28 ms. Nevertheless, the results show very promising savings and efficiency from FPGA mapping with this dynamic reconfiguration process.

[7] (2018) is a second significant contribution addressing PR. Starting out from an initial statement on excessive reconfiguration times, which range from 100 to 150 ms in their application study of an eight-layer CNN for facial landmark detection, the authors describe a DSE process in which they consider overlapping reconfiguration of parts of the inference path with other parts that are executed in software. Pooling and activation function layers run therefore in software while they alternate software and hardware executions of convolutional layers. The approach allows reducing the overall execution time to a factor of 2.24 compared to pure software execution. Finally, among this class of PR, [7] is the sole work, up to our knowledge, that considers the critical partitioning problem inherent with PR. A formal DSE approach is defined to both compute the optimal configuration of each layer and decide how to split the network into chunks that will be dynamically programmed via PR. Application on Binary Neural Networks (BNNs), specifically the CNVW1A1 Binary Neuralnet from Xilinx, reports a clear potential for resource reduction against static FPGA allocation, but with highly varying degrees of performance - occupation tradeoffs and unclear performance penalty from large reconfiguration times.

In conclusion, reductive PR is the most interesting approach for reducing FPGA logic requirements and thus extending the efficiency of static FPGA acceleration. The potential is vast for CNNs because most part of their processing can be moved to hardware, and this grows very quickly with the size of the topology. However, two important conditions must be met: i) a methodical DSE approach allowing the identification of suitable solutions in the very large and complex design space involved with PR and ii) important optimization of reconfiguration processes are also mandatory. In the following, we investigate this by using i) a previously defined methodology for the efficient mapping on many-core reconfigurable accelerated systems allowing PR [11] and ii) a very fast optimized reconfiguration controller [10].

3. PARTIAL RECONFIGURATION FLOW

In this study, we investigate the potential of using PR to improve the energy efficiency of CNNs on FPGAs (Fig. 1). A SystemC simulation methodology for modeling, scheduling, and estimation of multiprocessor reconfigurable systems called FoRTReSS [11] is used to explore the mapping of CNNs on reconfigurable platforms supporting PR. In this approach, applications are represented as Control Data Flow Graphs (CDFGs) consisting of several tasks, each with multiple possible implementations that can be hardware-mapped to the reconfigurable area or software-mapped to processor cores.

Table 1. Software/hardware characterization of GTSRB application tasks on Zedboard

Task (i)	Execution unit (j)	$T_{i,j}$ (ms)	$P^{static}_{i,j}$ / $P^{idle}_{i,j}$ / $P^{run}_{i,j}$ (mW)	N^{slice} / N^{bram} / N^{dsp}
img_load	Core	0.730	45 / 8 / 257.7	–
conv1(5x5, 6)	Core	2.247	45 / 8 / 257.7	–
	RR	0.898	– / 42 / 67	1462 / 0 / 25
maxpooling1	Core	0.023	45 / 8 / 257.7	–
	RR	0.024	– / 41 / 2	90 / 0 / 0
conv2(5x5, 16)	Core	1.334	45 / 8 / 257.7	–
	RR	1.285	– / 42 / 62	450 / 2 / 25
maxpooling2	Core	0.009	45 / 8 / 257.7	–
	RR	0.008	– / 41 / 3	78 / 0 / 0
conv3(5x5, 120)	Core	0.331	45 / 8 / 257.7	–
	RR	0.066	– / 41 / 47	256 / 0 / 15
fc1(84)	Core	0.066	45 / 8 / 257.7	–
	RR	0.105	– / 41 / 2	30 / 0 / 1
fc2(43)	Core	0.025	45 / 8 / 257.7	–
	RR	0.041	– / 41 / 2	21 / 0 / 1
softmax	Core	0.010	45 / 8 / 258	–

To identify relevant implementation solutions in this very large design space, it is crucial to consider execution time and resource consumption of each task, but also power, energy, and reconfiguration times. Hardware implementations are defined by considering actual task resources and performance, which can be based on HLS and lower-level estimates from FPGA synthesis tools. The target architecture is described by detailed FPGA resource organization and a set of processor cores. Using the characterization of the application tasks and the target platform architecture, FoRTReSS can automatically explore all possible RRs and finally produce a set of mappings fully scheduled on different cores and RRs. FoRTReSS provides energy consumption estimates for the full application and for software and hardware implementations of each task as well, allowing decisions on optimal energy efficiency at different levels.

In the following, we introduce the approach for modeling CNN applications and target platforms, as well as the exploration of design space by FoRTReSS. Specifically, in Section III.C and Section III.D, we provide an example showing the characterization and exploration results of a GTSRB (German Traffic Sign Recognition Benchmark, 32x32 color traffic sign images) CNN topology.

A. High-level Synthesis

The definition and simulation of the original CNN topology, also producing learning weights and bias values, are processed using Keras. Pure synthesizable C code has been developed to help fast prototyping of ConvNets and comprehensive hardware/-software deployment on FPGA following an HLS methodology of Xilinx. A 16-bit quantization scheme (Q8.8) is first applied currently to the reference 32-bit floating-point weight and bias values to produce a fixed-point model. The subsequent layer-based implementation supports full hardware implementation of 2D convolutional, max-pooling, and fully connected layers. Other processing functions such as file reading, pre-processing and post-processing of images are kept in software. This approach allows the creation of a wide range of popular network topologies, with quasi-automated RTL design ensuring complete practical implementation and execution on real FPGA platforms. Intra-layer parallelism is explored in terms of loop unrolling, pipelining, and array partitioning.

For each layer of the network, we explore loop-level parallelism which remains a manual process in the RTL design flow. Within the entire network architecture, convolutional layers are the most computationally demanding layers. The 2D convolution is based on six nested loops processing output channels, output feature maps (rows and columns), input channels, and K*K convolution kernels. Vivado HLS offers different pragmas to optimize the scheduling and resource allocation within these nested loops. A pipeline directive at the input channel level gives the best tradeoff between performance and on-chip resources. The inner nested loops are automatically unrolled by Vivado HLS, provided concurrent read/write accesses to the different arrays are possible. The input arrays are therefore partitioned adequately to let them to be stored in different BRAMs.

B. System Modeling

All the measurements and estimations described in the following constitute the necessary elements of an abstract model of the complete system, which can be further used by FoRTReSS DSE to process fast and reliable CNN mapping analysis and scheduling simulations.

B.1. Layer-wise processing functions:

For the CNN model in FoRTReSS, each layer is treated as an individual task, which may have a hardware or a software implementation. For hardware implementations, area and power are estimated for each task using Xilinx post-synthesis estimators. The power estimated for hardware implementations includes two components: idle and running power. Idle power represents the power needed by a task when configured in a RR but not actively executing. Running power refers to the additional power consumed during task execution within the RR. Execution times are derived from Vivado HLS reports. For software implementations, power characterization of a task is based on the power

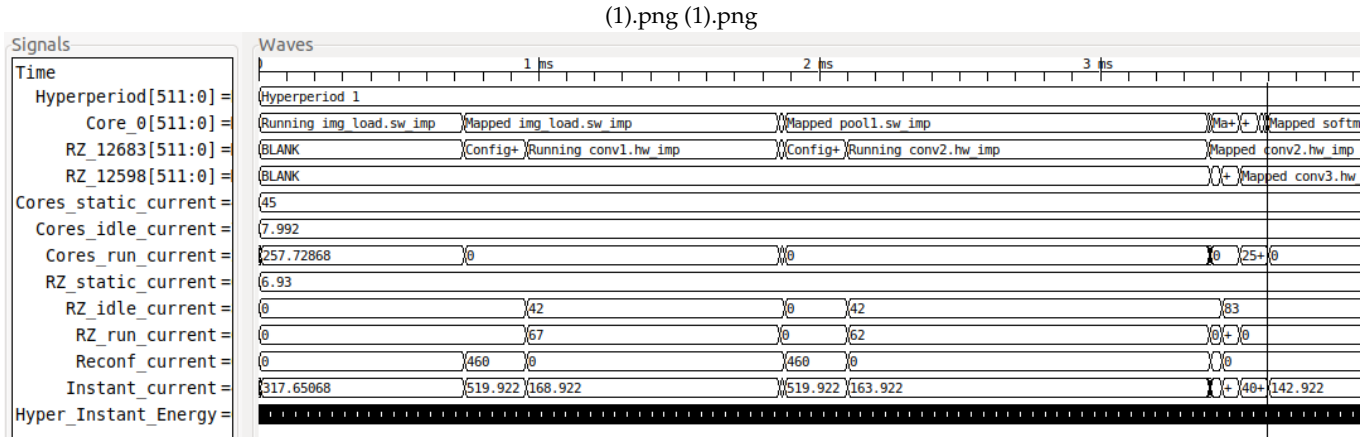

Fig. 2. Scheduling simulation results for GTSRB PR execution with two RRs

model of the Processing System (PS) of the platform. Execution times are derived from real measurements on one CPU core.

B.2. Reconfigurable platform:

This work targets Zynq series, specifically with a ZedBoard (dual-core Arm Cortex-A9, XC7Z020-CLG484-1) from the Zynq-7000 family and a ZCU102 (quad-core Arm Cortex-A53, XCZU9EG-FFVB1156-1) from the Zynq Ultrascale+ family, both of which support Dynamic Partial Reconfiguration (DPR). For the target reconfigurable platforms considered, modeling is usually divided into two distinct sections: the PS and the Programmable Logic (PL). The power model for PS is composed of three parts: static, idle, and active. The construction of this power model refers to the processor power model presented in [8], which can be generally expressed by the following formula:

$$P_{cpu} = P_{cpu}^{static} + \alpha * P_{cpu}^{idle} + \beta * P_{cpu}^{run} \quad \textbf{(1)}$$

where α and β are the coefficients expressing the idle and running contribution respectively for a given type of CPU.

Concerning the PL, we model the amount and arrangements of the different resources (logic cells, RAM blocks, DSP blocks) of the entire FPGA using XML files. For the set of RRs generated by FoRTReSS, the power model is similarly divided into three components: empty, idle, and running [9]. The empty (static) power corresponds to the power consumption of a region when it is unoccupied, which is estimated proportionally to the full static power consumption of the FPGA device, depending on the FPGA resources within the region. Idle and running power are associated with the task when it is configured and executed, respectively, on the corresponding region. Each component of the power model mentioned above is derived from Xilinx Power Estimator after logic synthesis.

Additionally, when implementing hardware with dynamic reconfiguration, it is fundamental to address reconfiguration efficiency. Reconfiguration times are excessive using original Xilinx controllers (ICAP, PCAP) which affects both performances and energy very significantly. Our approach to this well-known DPR problem is based on using the currently known best-performance reconfiguration controller in the literature, UPaRC (Ultra-fast Power-aware Reconfiguration Controller [10]), which can reach a reconfiguration throughput of 1.433GB/s. The reconfiguration controller is mainly characterized by two parameters, P_{reconf} and T_{reconf}, which characterize respectively the associated reconfiguration power and performance. P_{reconf} is extrapolated from [10] to consider the maximum operating frequency of 362.5 MHz (thus estimated at 460 mW). It is worth noting that due to its relatively small size (around 1000 slices), the associated idle power is set to zero. Reconfiguration time is estimated as follows:

$$T_{reconf} = \frac{Bitfile\ size}{Reconf\ controller\ throughput} \quad \textbf{(2)}$$

where the size of the bitfile can be calculated by FoRTReSS, by evaluating the quantity of resources of the RR. Similarly to the hardware static power, the reconfiguration time of a region depends on the size of that region, implying that smaller reconfiguration regions should be selected.

B.3. Application to GTSRB / ZedBoard:

The following details an example of CNN characterization. The application is a seven-layer convolutional neural network targeting GTSRB on ZedBoard. Key features of the implementation model include task id i, execution unit j (core or RR), execution time $T_{i,j}$, power $P_{i,j}$, and the number of FPGA resources N^{slice}, N^{dsp}, and N^{bram} for hardware implementations. Each task encompasses one or more implementations, with at least one software implementation.

As illustrated in Table 1, the hardware acceleration effect is significant for convolutional layers, especially for conv1 and conv3. In cases where energy consumption is also advantageous and depending on other scheduling constraints, these hardware implementations will be considered in the first place for execution. Conversely, for layers with approximately equal execution time on software and hardware (pool1, pool2) or faster execution on software (fc1, fc2), also considering the reconfiguration overhead, the allocation would be made preferably on a software core.

C. Design Space Exploration

C.1. System scheduling:

FoRTReSS generates and simulates full system mappings and associated schedulings for various possible deployments of the application /platform model. An illustrative example of this is provided in Fig. 2, which presents one implementation solution for the GTSRB topology using two RRs.

This scheduling trace illustrates the detailed and actual allocation, execution, and reconfiguration of each individual task. It additionally shows power and energy consumption at each scheduling event, for possibly several hyperperiods. Table 3

Table 2. Efficiency comparison of software and hardware (static and PR) implementations for MNIST, GTSRB (Zedboard) and CIFAR-10 (ZCU102)

Implementation	Execution time (ms)	Average Power (mW)	Energy consumption (mJ)
MNIST SW	11.565	310.7	3.593
MNIST Static HW	**2.205**	468.5	**1.033**
MNIST PR HW	4.466	458.8	2.049
GTSRB SW	4.775	310.4	1.482
GTSRB Static HW	**3.112**	357.3	1.112
GTSRB PR HW	3.594	256.5	**0.910**
CIFAR-10 SW	198.697	220.1	43.734
CIFAR-10 Static HW	**14.078**	1341.2	18.882
CIFAR-10 PR HW	16.535	682.0	**11.277**

details values of energy, performance, and power per each CNN layer, from GTSRB PR scheduling simulation with two RRs on ZedBoard.

C.2. RR partitioning:

Fig. 3 illustrates the placement of two RRs as part of a PR implementation solution for the GTSRB topology on Zedboard. FoRTReSS defines several candidates RRs based on the resources available on the device and the characterization of tasks. To make the best choices, these RRs are further evaluated using a cost function. This takes into account how complex their shape is, how many tasks they can handle, and how much of their resources are left unused, known as internal fragmentation [11]. This careful evaluation helps in defining and picking the most relevant regions for the tasks and identifying the best FPGA partitioning for the global application.

4. VALIDATION STUDY

In the following, we address the design space exploration of three typical CNN benchmarks by comparing their software and hardware implementations (static and PR): MNIST (Modified National Institute of Standards and Technology database, 28x28 grayscale handwritten digits), GTSRB and CIFAR-10 (Canadian Institute for Advanced Research, 32x32 color object and animal images). We consider three standard topologies associated with these benchmarks, that are further trained and validated using Keras to serve as a reference for accuracy and provide the weights and biases used for CNN inference in the corresponding HLS C code. For the topologies associated with MNIST and GTSRB, we analyze different implementations (software, static hardware, PR hardware) on ZedBoard using FoRTReSS. For CIFAR-10 we consider a ZCU102 platform since ZedBoard is too small. For PR hardware, we only report the best energy-efficient solution with the optimal number of RRs.

On top of these three benchmarks, we also address a short comparison with existing works related to LeNet-5 on XC7Z020, in a way to better assess the relevance of numbers and conclusions reported.

A. CNN Benchmarks Exploration Results

A.1. MNIST:

The first benchmark is a five-layer CNN applied to the MNIST dataset (improved LeNet-5), composed of two convolutional layers, two max-pooling layers, and two fully connected layers (28x28-20C5-P2-40C5-P2-400-10). The corresponding classification accuracy is 99.05%. The best energy-efficient solution for PR HW identified by FoRTReSS DSE is based on using two RRs.

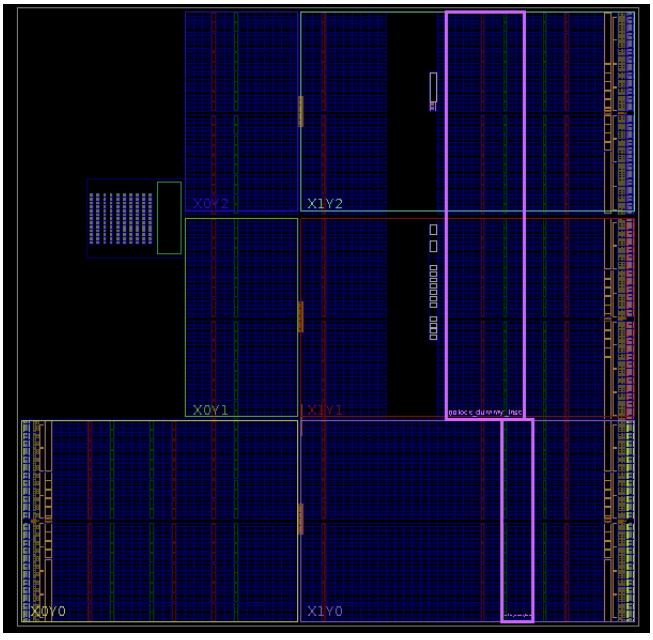

Fig. 3. Layout of RRs placement for the solution HW PR with two RRs for GTSRB topology on Zedboard

For the MNIST dataset, both hardware implementations perform better in energy consumption than software (Table 2). PR HW reports 2.59 times faster and 1.75 times better energy efficiency compared to software. However, due to reconfiguration times, PR is 2.03 times slower than static hardware, particularly

Table 3. Power and performance breakdown for GTSRB PR execution on Zedboard

Task (i)	Energy (mJ)	Execution time (us)	P_core (mW)[1]	P_RZ (mW)[1]	P_Reconf (mW)	P_total (mW)
img_load	0.230	730	45 + 8 + 257.7	6.93 + 0 + 0	0	317.7
config_conv1	0.116	222 (RZ 1)	45 + 8 + 0	6.93 + 0 + 0	460	519.9
running_conv1	0.152	898	45 + 8 + 0	6.93 + 42 + 67	0	168.9
running_pool1	0.007	23.1	45 + 8 + 257.7	6.93 + 42 + 0	0	359.7
config_conv2	0.117	222 (RZ 1)	45 + 8 + 0	6.93 + 0 + 0	460	519.9
running_conv2	0.211	1285	45 + 8 + 0	6.93 + 42 + 62	0	163.9
running_pool2	0.002	8.98	45 + 8 + 257.7	6.93 + 42 + 0	0	359.7
config_conv3	0.022	38.3 (RZ 2)	45 + 8 + 0	6.93 + 42 + 0	460	561.9
running_conv3	0.013	66	45 + 8 + 0	6.93 + 83 + 47	0	189.9
running_fc1	0.026	65.9	45 + 8 + 257.7	6.93 + 83 + 0	0	400.7
running_fc2	0.010	24.9	45 + 8 + 257.7	6.93 + 83 + 0	0	400.7
softmax	0.004	10	45 + 8 + 257.7	6.93 + 83 + 0	0	400.7
Total	**0.910**	**3594**	-	-	-	-

[1] P = P_static + P_idle +P_run

for the conv2 layer, as the associated RR is quite large, covering nearly half the size of the FPGA device. The corresponding reconfiguration time ends up representing around 25% of the total CNN inference time, also negatively impacting the energy consumption of the PR solution which is 1.98 times less energy-efficient than static hardware.

It is worth noting that in terms of average power, and despite previous net energy benefits, both static and PR HW consume more than software (468.5 mW and 458.8 mW, vs. 310.7 mW). This is due to the relatively high static power consumption of FPGA devices compared to hard IP cores on Zynq. In terms of resource occupation, Table 4 reports a slight increase of Slices and DSPs for PR HW, and around 2x more BRAM compared to static HW. This can be attributed to the fact that the total resources covered by the rectangular RR generated for conv2 marginally exceed those of the static implementation.

A.2. GTSRB:

The topology used with the GTSRB dataset consists of three convolutional layers, two max-pooling layers, and two fully connected layers (32x32-6C5-P2-16C5-P2-120C5-84-43). Classification accuracy is 85.20% for this benchmark. The best energy-efficient solution for PR HW identified by FoRTReSS DSE is based on using two RRs.

Like previously on the MNIST dataset, hardware solutions have better performance and energy efficiency compared to software execution (Table 2). PR HW brings 1.33 times (24.7%) better inference time and 1.63 times (38.6%) higher energy efficiency compared to software. In terms of execution time, PR HW remains slightly slower (+15.5%) than static HW due to the additional reconfiguration times.

However, in contrast to the MNIST benchmark, PR HW is 1.22 times (18.2%) more energy-efficient than static implementation. With more layers, there is also more potential for hardware acceleration and better efficiency. In addition, for this benchmark PR HW saves more resources compared to static HW (Table 4), resulting in a more significant improvement of 28% in average power consumption (256.5 mW vs. 357.3 mW). Comparing with software (310.4 mW) further highlights the benefits of PR on this example, but also the relatively high cost of static power inherent to static HW implementations.

A.3. CIFAR-10:

The last benchmark, which is also the largest topology, consists of six convolutional layers, three max-pooling layers, and one fully connected layer (32x32-32C3-32C3-P2-64C3-64C3-P2-128C3-128C3-10). It is applied to the CIFAR-10 dataset with a classification accuracy of 79.2%. The best energy-efficient solution for PR HW identified by FoRTReSS DSE utilizes two RRs.

For this dataset, as demonstrated in Table 2, PR HW continues to report faster performance and lower energy consumption than software. It is notably 12.02 times (91.7%) faster and 3.88 times (74.2%) more energy-efficient, indicating a more pronounced improvement compared to the two previous topologies. Similar to the GTSRB benchmark, the PR implementation is slightly slower (+17.5%) than static HW due to reconfiguration overheads.

It should be noted here that the CIFAR benchmark, the largest CNN example in this application study, is where PR has the best results against static implementation with 1.67 times (40.3%) more energy efficiency. In this benchmark, PR HW provides the most important FPGA resource savings among the three benchmarks examined as well (Table 4). Similar to MNIST and GTSRB, PR HW occupies more BRAM resources than static HW. This is primarily due to the fact that the tasks within this topology utilize fewer BRAMs. However, FoRTReSS generates rectangular RRs that occupy a full column of BRAMs. As a consequence, additional BRAMs are present and contribute to the total increased count. Conversely, it achieves approximately 40% savings for Slice and DSP resources. This results in better average power consumption for PR against static HW with 49% improvement.

Unlike GTSRB, here PR HW is less power efficient than software (682.0 mW vs. 220.1 mW). This is due to the size of the network, featuring more layers, and leading to increased resource usage and higher static power in comparison to GTSRB. Static HW is not very efficient for the same reason, consuming up to six times more average power than software for this relatively large topology.

In light of these results, it appears that PR can notably improve the processing efficiency of static hardware acceleration

Table 4. Resource occupation of hardware implementations (static and PR) for MNIST, GTSRB (Zedboard) and CIFAR-10 (ZCU102)

Implementation	Functions on HW	N^{slice}	N^{bram_18k}	N^{dsp}
MNIST Static HW	conv1, conv2, fc1	**6764** / 13300	99 / 280	136 / 220
MNIST PR HW (2 RRs)		6900 / 13300	220 / 280	140 / 220
GTSRB Static HW	conv1, conv2, conv3	2168 / 13300	2 / 280	65 / 220
GTSRB PR HW (2 RRs)		**1900** / 13300	20 / 280	60 / 220
CIFAR-10 Static HW	conv2, conv3, conv4, conv5, conv6	28408 / 34260	6 / 1824	777 /2520
CIFAR-10 PR HW (2 RRs)		**18360** /34260	48 / 1824	480 / 2520

for CNNs, provided certain conditions are met. This is clearly the case for the two largest topologies investigated, GTSRB and CIFAR-10, with respectively 1.22 (18.2%) and 1.67 times (40.3%) more energy efficiency than static hardware. However, for MNIST which is the smallest topology, fewer layers means less hardware acceleration and RR reuse potential, making PR improvements less impactful.

B. Comparison with Other Works

In this section, we attempt to compare our results to other works. It should be noted here that it is difficult to identify some works for this, with so many design options: CNN benchmark and topology, FPGA device, and implementation parallelism. Most works report on actual performances, among which few allow for power and energy comparison, also considering both static and dynamic solutions. In the following, we have been able to set up an interesting comparison of the MNIST benchmark with fpgaConvNet [6], in terms of performance. Indeed the paper reports a measured performance of 0.48 GOp/s for a LeNet-5 CNN (32x32-6C5-P2-16C5-P2-120-84-10, 98.57%) on an XC7Z020 device at 100 MHz. [13] (2019) is another work addressing the implementation of LeNet-5, but there is no PR involved. However, they report comparable performance with 0.343 GFLOPS on the ZYBO Z7 board (XC7Z020) at 100 MHz.

Our results on the MNIST (28x28-20C5-P2-40C5-P2-400-10, 99.05%) on an XC7Z020 device at 100 MHz, despite higher topology complexity, reflect a similar level of performance with 0.83 GOp/s for the static implementation and 0.41 GOp/s for the dynamic implementation. In other words, we are at the same level of performance with [6] but for a twice bigger topology with approximately 50% fewer resources for the dynamic solution (cf. Table 4). This would allow to target smaller devices for implementation, thus highly reducing power consumption when considering that MNIST is the least energy efficient of the three benchmarks investigated.

5. CONCLUSION AND PERSPECTIVES

In this work, we proposed a novel and elegant approach to improve the efficiency of CNN inference on FPGAs exploiting reductive PR. This approach partitions the original CNN topology based on various layers and reconfigures the PL with a smaller number of reconfigurable resources at runtime. This methodology greatly helps to explore automatically different FPGA partitioning and arrangements of RRs and to identify more energy-efficient hardware/software mapping solutions given the extended size of the PR codesign space.

We assessed this approach on three different network examples, showing potential improvements reaching a factor of 3.88 (gain vs. software solution) and 1.67 (gain vs. static solution) in energy savings. The improvement levels greatly depend on the inner potential of layers for hardware acceleration and grow steadily with the depth of the network as more hardware reuse is possible.

From these results, future works will investigate an application study on deeper, widely-used CNN networks, such as ResNet50. Ongoing works also address the automated implementation of the identified solutions on real platforms following the methodology of [12] on Xilinx Zedboard and ZCU102 platforms.

REFERENCES

1. M. Farhadi, M. Ghasemi, Y. Yang, “A Novel Design of Adaptive and Hierarchical Convolutional Neural Networks using Partial Reconfiguration on FPGA", IEEE High Performance Extreme Computing Conference (HPEC), Waltham, MA, USA, 2019, pp. 1-7, doi: 10.1109/HPEC.2019.8916237.
2. E. Youssef, H. A. Elsimary, M. A. El-Moursy, H. Mostafa and A. Khattab, “Energy-Efficient Precision-Scaled CNN Implementation With Dynamic Partial Reconfiguration", in IEEE Access, vol. 10, pp. 95571-95584, 2022, doi: 10.1109/ACCESS.2022.3204704.
3. E. Youssef, H. A. Elsemary, M. A. El-Moursy, A. Khattab and H. Mostafa, “Energy Adaptive Convolution Neural Network Using Dynamic Partial Reconfiguration", 2020 IEEE 63rd International Midwest Symposium on Circuits and Systems (MWSCAS), Springfield, MA, USA, 2020, pp. 325-328, doi: 10.1109/MWSCAS48704.2020.9184640.
4. H. Irmak, D. Ziener and N. Alachiotis, “Increasing Flexibility of FPGA-based CNN Accelerators with Dynamic Partial Reconfiguration", 2021 31st International Conference on Field-Programmable Logic and Applications (FPL), Dresden, Germany, 2021, pp. 306-311, doi: 10.1109/FPL53798.2021.00061.
5. C. Alberto de Albuquerque Silva, A. Andrey Ramalho Diniz, A. Duarte Dória Neto, J. Alberto Nicolau de Oliveira, “Use of Partial Reconfiguration for the Implementation and Embedding of the Artificial Neural Network (ANN) in FPGA", 4th International Conference on Pervasive and Embedded Computing and Communication Systems, 2014, doi: 10.5220/0004716301420150.

6. Stylianos I. Venieris and Christos Savvas Bouganis, “fpgaConvNet: A Framework for Mapping Convolutional Neural Networks on FPGAs", 2016 IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM), Washington, DC, USA, 2016, pp. 40-47, doi: 10.1109/FCCM.2016.22.
7. F. Kastner, B. Janben, F. Kautz, M. Hubner, G. Corradi, “Hardware/Software Codesign for Convolutional Neural Networks Exploiting Dynamic Partial Reconfiguration on PYNQ", 2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), Vancouver, BC, Canada, 2018, pp. 154-161, doi: 10.1109/IPDPSW.2018.00031.
8. R. Bonamy, S. Bilavarn, F. Muller, F. Duhem, S. Heywood, P. Millet, F. Lemonnier, “Energy efficient mapping on manycore with dynamic and partial reconfiguration: Application to a smart camera", International Journal of Circuit Theory and Applications, Wiley, 2018, doi: 10.1002/cta.2508.
9. 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.
10. 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.
11. 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.
12. A. Sadek, A. Muddukrishna, L. Kalms, A. Djupdal, A. Podlubne, D. Goehringer and M. Jahre, “Supporting Utilities for Heterogeneous Embedded Image Processing Platforms (STHEM): An Overview", International Symposium on Applied Reconfigurable Computing (ARC), 2018, doi: 10.1007/978-3-319-78890-6_59.
13. D. Rongshi and T. Yongming, “Accelerator Implementation of Lenet-5 Convolution Neural Network Based on FPGA with HLS", 2019 3rd International Conference on Circuits, System and Simulation (ICCSS), Nanjing, China, 2019, pp. 64-67, doi: 10.1109/CIRSYSSIM.2019.8935599.