Exploring NEURAGHE: A Customizable Template for APSoC-based CNN Inference at the Edge

Exploring NEURAGHE: A Customizable Template for APSoC-based CNN Inference at the Edge

Paolo Meloni, Daniela Loi, Gianfranco Deriu, Marco Carreras, Francesco Conti, Alessandro Capotondi and Davide Rossi

***Abstract*—The NEURAGHE architecture has proved to be a powerful accelerator for Deep Convolutional Neural Networks running on heterogeneous architectures based on Xilinx Zynq-7000 APSoCs. NEURAGHE exploits the processing system and the programmable logic available in these devices, to improve performance through parallelism and to widen the scope of use-cases that can be supported. In this work, we extend the NEURAghe template-based architecture to guarantee design-time scalability to multi-processor SoCs with vastly different cost, size and power envelope such as Xilinx's Z-7007s, Z-7020 and Z-7045. The proposed architecture achieves state-of-the-art performance and cost effectiveness in all the analyzed configurations, reaching up to 335 GOps/s on the Z-7045.**

I. INTRODUCTION

Convolutional Neural Networks (CNNs) are one of the most promising classes of deep learning algorithms due to remarkable performance achieved in a broad area of applications, ranging from speech recognition to computer vision and natural language processing [1]. However, improvements in the accuracy of a CNN come at the expense of increased computational and memory workload. The execution of CNN algorithms involves a huge number of Multiply Accumulate (MAC) operations representing the core of the convolution kernels, and requires significant memory storage and bandwidth for storing and accessing weights efficiently. Therefore, there is a growing need for low-cost hardware platforms capable of running computationally intensive CNN-based applications fast and efficiently. Due to the intrinsic parallelism in such convolutions, Field-Programmable Gate Arrays (FPGA) are a very promising target technology for the implementation of hardware accelerators for different kind of neural networks [2], [3], [4], [5]. Exploiting dedicated hardware, FPGAs allow to realize very flexible architectures. They can integrate a high number of parallel Digital Signal Processing (DSP) units, usable to efficiently implement MAC operations and to deliver high throughput at limited clock frequencies. Moreover, FPGAs offer a significant amount of memory resources, which can be used to create temporary storage buffers for partial convolution results, and other hardware primitives, such as registers and look-up tables (LUTs), suitable to implement the glue logic around the accelerators. Based on these considerations, in [6] we have proposed NEURAGHE, a programmable CNN accelerator exploiting the synergistic execution on ARM processor and on the programmable logic of Xilinx Zynq Z-7000 All Programmable System-on-Chip (APSoC). Automated generation of CNN processors, based on High-level synthesis or data-flow generators, has often been argued to be more effective than a template-based design in maximizing the utilization of the available FPGA resources [2], [7]. Contrary to this line of thought, in this work we show how the NEURAGHE architecture can be tailor-cut to support different trade-off optimization scenarios with respect to cost, computation efficiency and power consumption, and to meet a wide scope of use-cases requiring at-the-edge CNN inference. We demonstrate that, thanks to its unique flexibility, NEURAGHE can fit on a wide scope of devices, ranging from low-end IoT nodes to mid-to-high end embedded processing platforms. NEURAGHE outperforms all existing architectures on Z-7045 using 8-bit and 16-bit data (delivering up to 335 GOps/s or 173 GOps/s, respectively), achieves state-of-the-art performance on Z-7020 using 8-bit data (up to ∼85 GOps/s), and can be used even on tiny devices such as Z-7007s.

P. Meloni, D. Loi, G. Deriu and M. Carreras are with DIEE, University of Cagliari, Cagliari, 09123 Italy.

F. Conti, A. Capotondi and D. Rossi are with DEI, University of Bologna, Bologna 40126 Italy.

Manuscript received XX YY, 20XX; revised XX YY, 20XX.

This work has received funding from the European Union's Horizon 2020 Research and Innovation Programme under grant agreement No. 780788

II. NEURAGHE ARCHITECTURAL TEMPLATE

The NEURAGHE[1] architecture consists of a hierarchical structure that overlaps with the hardware organization of Xilinx Zynq SoCs (adaptable to other APSoCs on the market). It contains a General-Purpose Processor (GPP), i.e. the ARM-based Processing System (PS) in the Zynq, a memory-mapped off-chip DDR (as available on the Zynq), and a set of Convolution Specific Processors (CSP), hosted on the Programmable Logic (PL), acting as accelerator. Fig. 1 illustrates the overall system-level organization of the NEURAGHE architecture.

NEURAGHE's CSPs are designed to operate the requested convolutions autonomously, relieving the GPP from any control-related action. The design enables a synergistic usage of GPP to take care of effective computing workloads, such as e.g. data marshaling or linear layers, while CSPs elaborate convolutions. This execution model is available through a dedicated software library provided with NEURAGHE [6], optimized to exploit the NEON extension and the two cores available on the ARM system. Each Convolution Specific Processor is composed of: i) a RISC microcontroller (μC) executing a control firmware for synchronizing transfers and convolution computations; ii) a Convolution Engine (CE) that represents the computational core of the accelerator, with a

[1]The name of the accelerator template derives from the ancient megalithic edifices named *nuraghes*, typical of the prehistoric culture in Sardinia. The different configurations are named after most important *nuraghes*. https://en.wikipedia.org/wiki/Nuraghe

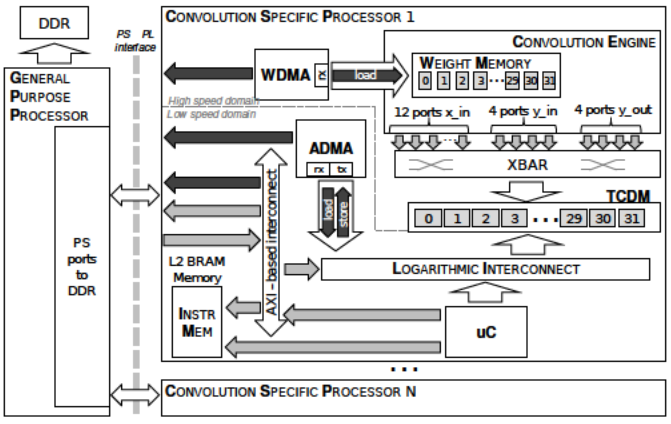

Fig. 1. NEURAGHE overall architectural template.

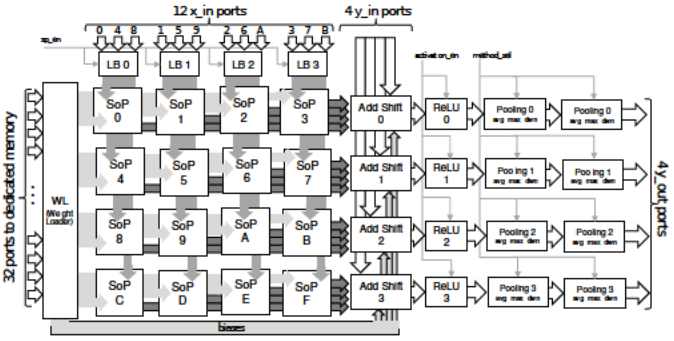

Fig. 2. NEURAGHE CE internal organization with 4×4 SoP matrix.

private weight memory and several ports to read the input pixels and write the results of convolution; iii) a pair of Direct Memory Access (WDMA and ADMA), respectively dedicated to weights and to activations; iv) a Tightly Coupled Data Memory (TCDM) to store the activations, and an instruction memory to store the code for the RISC microcontroller.

The Convolution Engine embeds a set of Sum-of-Products (SoP) units, used to deploy convolutions on the reconfigurable logic, which are organized as a matrix of N columns and M rows. The SoP matrix is the most resource-consuming component and defines the number of input feature maps and output feature maps processed in each iteration. Fig. 2 shows the internal organization of the architecture, highlighting the composition of the CE module when a SoP matrix with 4 columns and 4 rows is used. In each cycle of activity, the CE collects up to 12 input features and computes their contributions to 4 output features. The input features are loaded through a set of line buffers, LB in the diagram, used to cache few lines of the input image. In this way, by loading a single new pixel per cycle, an entire new window of the image can be dispatched to the SoP matrix for convolution. Considering 16-bit pixel data, each LB is fed with two pixels obtained from the input port and produces two square convolution windows per cycle. The convolution windows are then consumed by the SoP units. To map efficiently on the FPGA DSP resources, the SoP units can be configured at design time to be partitioned in multi-trellis structures of multiply and addition operations, whose outputs are summed together using a dedicated adder. The output pixels produced by each Adder-shifter module are then streamed into a block that, when enabled, performs Rectifier Linear Unit (ReLU) activation function. The CE integrates also a pooling layer, that temporarily stores and compares output pixels in square pooling windows, to select on-the-fly one single output pixel per window, according to the selected operating mode (max pooling, average pooling and simple down-sampling). To reduce the power consumption, the CPS clock domain is partitioned in two regions. An high-speed domain, only reserved for throughout-critical components, such as the CE and the WDMA, and a low-speed domain reserved for the rest of the circuitry.

III. Design-Time Configuration Parameters

In this work, we have extended the architecture presented in [6] to support different configuration parameters, to fit on different heterogeneous SoCs and to optimize the performance of specific networks on the same target. In the following sections, a description of each parameter is provided:

a) Number of Convolution Specific Processors: NEURAGHE can host multiple Convolution Specific Processors that can be used in parallel to map independent layers.

b) Convolutional Engine SoP matrix size: The SoP matrix is the main scalable computing datapath of each Convolution Engine. Designers can change the size of the matrix acting on the number of rows and columns that it contains, creating configurations that fit devices featuring different number of DSP slices. The number of rows and columns in the matrix defines, respectively, the number of IF maps consumed by the engine and the number of OF maps produced by it.

c) Convolutional Engine arithmetic precision: The designer can choose to set data precision to 16-bit or 8-bit, to trade off accuracy for performance. All data (i.e. input pixels, output pixels, biases and weights) will be represented with the same number of bits. It is also possible to instantiate support for run-time selection of the data precision.

IV. Hardware Implementation

Thanks to its flexibility, the NEURAGHE template can be reshaped to fit on a wide range of APSoC devices, to conveniently select the best performance/power/cost trade-off for a specific use-case. To assess the approach, we have taken as reference the Xilinx Zynq-7000 family. We have tested implementation on three different Zynq SoC devices, easily accessible on commonly available development boards, which integrate an ARM-based processing system with different sizes of FPGA circuitry: the Zynq Z-7045, the Zynq Z-7020 and the Zynq Z-7007s. These targets cover nearly the whole range of devices in the family, suitable to be used in embedded applications. As shown in Table I, we chose three configurations of the NEURAGHE architecture fitting respectively in the three target devices: LOSA (4×4 SoP matrix size), SABINA (2×2 SoP matrix size), and BANZOS (1×1 SoP matrix size). SoP matrix sizes have been chosen to maximize the usage of the DSP slices available on the chip (at least 82% utilization on all the configurations). Moreover, we have implemented ARRUBIU, that fits on the Z-7045 SoC but has a different CSP organization, featuring two CSP with a 2×4 CE matrix per CSP. Considering that every CSP uses two High-Performance PS-to-PL ports and that four ports are available in Zynq devices, only one or two CSPs can be instantiated on this family of SoCs.

TABLE I
RESOURCES UTILIZATION OF THE DIFFERENT CONFIGURATIONS.

	LOSA	ARRUBIU	SABINA	BANZOS
Target SoC	Z-7045	Z-7045	Z-7020	Z-7007s
#CSPs; SoP	1; 4×4	2; 2×4	1; 2×2	1; 1×1
DSP	864 (96%)	864 (96%)	216 (98%)	54 (82%)
LUTs	99546 (51%)	101352 (46%)	43880 (83%)	12610 (87%)
BRAMs	320 (59%)	288 (53%)	136 (97%)	44 (88%)

V. PERFORMANCE EVALUATION

Table II reports performance and efficiency levels achievable by the four configurations on several benchmarks, in terms of actual computational power (GOps/s), energy efficiency (GOps/sec per Watt), and price efficiency (GOps/sec per k$).

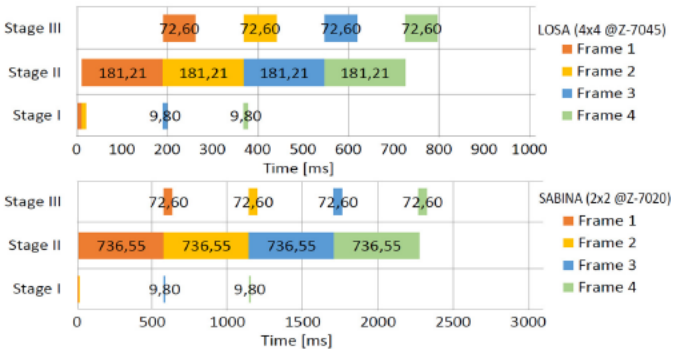

Fig. 3. VGG-16 four frame execution timeline. Bars indicate execution time of the different stages, expressed in *ms*. Stage I consists on input data preparation, executed by the GPP. Stage II consists of all the computational blocks executed on the CSP. Stage III consists all the rest, mainly fully-connected layers, executed on the GPP. Execution stages related to different frames can overlap.

As previously mentioned, NEURAGHE exploits the interaction between the General Purpose Processor and the CSP. As an example, in Figure 3 you may notice how different execution stages on different image frames overlap with each other in a task-level pipeline fashion. Such a parallel approach can be exploited, with different speed-up, on all the different target devices. In general, processing tasks assigned to the CSP, which are the limiting nodes of the pipeline, are those that are impacted by the scaling to different technology targets.

A. Exploring SoP matrix size

Sum-of-Products matrix size can be selected to optimally exploit the usage of DSP slices available on the target. In general, while implementations on more expensive devices reach higher absolute performance levels, smaller configurations, e.g. SABINA, expose a higher efficiency of SoP utilization with respect to peak performance. This can be expected, as shown by the roofline models of LOSA and SABINA plotted in Figure 4. Smaller configurations require lower levels of operational intensity to operate at their peak performance, limited by computing resources. For example, in Figure 4, the performance level achievable for the Conv layer corresponding to the vertical line, is limited by the peak performance when using SABINA, while it is limited by the PS-to-PL bandwidth when using LOSA, determining an efficiency degradation. Thus, smaller matrices work more efficiently, especially in initial and final layers of common CNN models, that expose in general lower operational intensity (see Figure 5). Moreover, when the number of input features in a Conv layer is not multiple of the number of CSP input ports, the SoP matrix is under-utilized during the last round of computation. Reduced SoP matrix size can mitigate the chances of under-utilization.

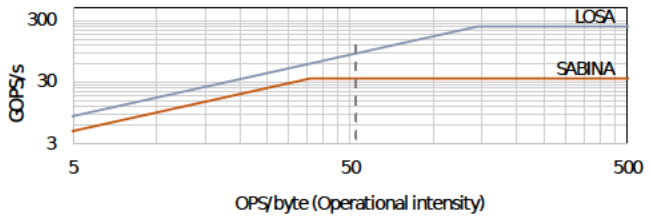

Fig. 4. Roofline model for LOSA and SABINA CSPs. The vertical axis represents the throughput, while the horizontal axis represents the operational intensity of the kernel. Vertical dashed line highlights the average operational intensity of a convolutional layer (∼51 Ops/B).

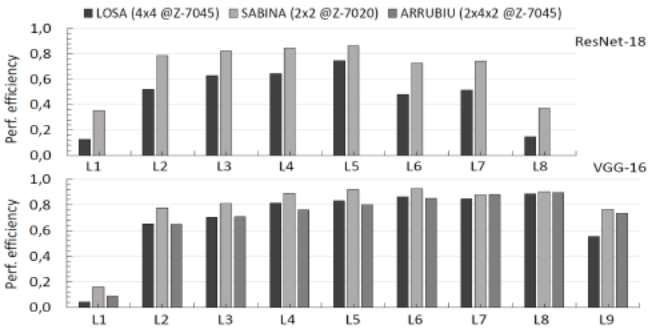

Fig. 5. Comparison of the performance efficiency, achieved on all the convolution layers (L) in ResNet-18 and VGG-16, by different NEURAGHE configurations. It is estimated as the ratio between actual performance (GOps/s) and peak performance (GOps/s). For VGG-16 we report also efficiency for a double-cluster configuration (ARRUBIU). We do not report CSP efficiency for ARRUBIU on ResNet-18 since the overall network efficiency is mainly limited by the GPP, see Section V-B.

Implementing more efficient configurations on low-cost hardware opens a whole range of possibilities, involving distributed processing and multi-board designs. For example, if allowed by the use-case, a designer may choose to replace a single LOSA with two SABINA configurations, to execute a task based on ResNet-18, achieving around 88% of the original performance, saving around 64% of costs.

B. Exploring number of CSPs

Multi-CSP configurations can be used to combine the benefits of a bigger device with those of a reduced matrix size. As an example, we have used the ARRUBIU configuration to run both VGG-16 and ResNet-18 benchmarks. We have tested two kinds of parallelism: *inter-frame parallelism*, that alternately sends input frames to each of the CSP, and *intra-frame parallelism*, that partitions input frames in tiles and alternately sends tiles to each of the CSP. On the other hand, any kind of feature-level parallel pattern significantly increases the amount of data to be transferred through the DMA causing huge performance drop. In general, on ARRUBIU *inter-frame parallelism* is more efficient. It leads to an increasing performance of around 10% for VGG-16 compared to LOSA (see Table II). On the other side, parallelizing by tiles is not so effective. In this case the CSPs have to operate

TABLE II
EVALUATION OF DIFFERENT NEURAGHE CONFIGURATIONS (* *indicates results based on inter-frame parallelism, the dual number indicates intra-frame*).

	LOSA single 4×4	ARRUBIU dual 2×4		SABINA single 2×2	LOSA single 4×4	ARRUBIU dual 2×4		SABINA single 2×2	BANZOS single 1×1	
Xilinx Zynq SoC (Price)	Z-7045 ($2500)	Z-7045 ($2500)		Z-7020 ($450)	Z-7045 ($2500)	Z-7045 ($2500)		Z-7020 ($450)	Z-7007s ($89)	
DSP [#]; Freq [MHz]	864; 140	864; 140		216; 120	864; 140	864; 140		216; 120	54; 80	
Benchmark net	ResNet-18	ResNet-18		ResNet-18	VGG-16	VGG-16		VGG-16	SqueezeNet	All-CNN-C
GOps/s (16-bit)	61.91	59.84*	53.52	27.51	172.67	188*	170.81	42.48	7.46	6.21
GOps/s per Watt (16 bit)	6.19	5.98*	5.35	7.86	17.26	18.8*	17.08	12.56	2.98	2.49
GOps/s per k$ (16 bit)	25	21.4*	23.93	61.13	69	75.2*	68.3	97.5	95	27.93
GOps/s (8-bit)	111.12	63.94*	73.27	49.61	335.09	370.44*	296.27	84.77	14.05	10.62

TABLE III
COMPARISON BETWEEN NEURAGHE AND OTHER ALTERNATIVES IN LITERATURE WHEN USING 8- OR 16-BIT ON VGG-16.

	This work	Venieris et al. [7]	Sharma et al. [8]	Guo et al. [9]	**This work**	Venieris et al. [10]	Guo et al. [9]
Xilinx Zynq SoC	**Z-7020**	Z-7020	Z-7020	Z-7020	**Z-7045**	Z-7045	Z-7045
Frequency [MHz]	**120**	125	150	214	**140**	125	150
Performance (16-bit) [GOps/s]	**42.48**	48.53	31.38	-	**172.67**	155.81	137
Performance (8-bit) [GOps/s]	**84.77**	-	-	84.30	**335.09**	-	292

on smaller features and performance overheads related mostly with loading of line buffers have higher impact. Moreover, input data tiles have to overlap with each other, generating an additional overhead in terms of repeated MAC operations to be payed when computing border pixels. Exploiting *inter-frame parallelism* is not possible in every use-case. For example, in ResNet-18, the GPP in the architecture is used very frequently along the CNN datapath for marshalling and shortcuts. Thus it cannot serve as companion of two different CSPs, without compromising the overall scheduling efficiency with respect to a single-CSP configuration like LOSA (see Table II).

C. Exploring CE arithmetic precision

In NEURAGHE configurations processing 16-bit activation and weights, each DSP slice in the programmable logic performs one MAC operation per cycle. NEURAGHE allows the programmer to select at runtime an operating mode processing 8-bit data, allowing to execute 2 MAC operations per cycle on each slice, prospectively improving performance by a factor of 2. As shown in Table II, speed-up measured in 8-bit operating modes is very close to the theoretical limit.

D. Comparison With S.o.A.

In Table III, we show a comparison, on VGG-16, between the proposed platform and some related work. The availability of design-time configurability and runtime tuning (selection between 8- and 16-bit data precision) makes NEURAGHE more flexible with respect to alternatives. On Z-7020, with respect to [7], NEURAGHE provides around the same performance, but supports more data types. The same results from comparison with [9], that does not provide results for 16-bit, and provides slightly lower performance for 8-bit. Another alternative in literature is [8]. NEURAGHE is more flexible, since [8] only supports 16-bit, and provides higher performance, being ∼35% faster. On Z-7045, NEURAGHE outperforms competitors for all considered data precision values. NEURAGHE is ∼10% faster of the work in [10] (20%, if we consider ARRUBIU) when using 16-bit, and ∼14% faster than the work in [9] for 8-bit. Moreover, to the best of our knowledge, NEURAGHE is the only alternative presenting an implementation on very low-cost platforms such as BANZOS (see rightmost column of Table II). This configuration is suitable to be used to build low-cost FPGA-accelerated IoT nodes. We report also performance achieved by BANZOS on a lightweight CNN [11], indicated as All-CNN-C, performing classification on low-resolution images. In this case, it supports ∼7 FPS at 16-bit precision and ∼13 FPS when using 8-bit.

VI. CONCLUSION

In this paper we present NEURAGHE, a customizable architecture for CNN acceleration on FPGA-endowed SoCs, focusing on its parametrization capabilities. Changing parameters of the NEURAGHE architecture leads to a variety of configurations that may be used in different use-cases to fit in different embedded systems, ranging from entry-level to high-end. This offers different trade-off optimization scenarios with respect to performance, cost and power consumption.

REFERENCES

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep Learning," *Nature*, vol. 521, pp. 436–444, 2015.

[2] M. Blott *et al.*, "FINN-R: An end-to-end deep-learning framework for fast exploration of quantized neural networks," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 11, no. 3, pp. 1–23, Dec. 2018.

[3] S. Mittal, "A survey of FPGA-based accelerators for Convolutional Neural Networks," *Neural Computing and Applications*, vol. 30, pp. 1–31, 2018.

[4] D. Pani *et al.*, "An FPGA platform for real-time simulation of spiking neuronal networks," *Frontiers in Neuroscience*, vol. 11, p. 90, 2017.

[5] C. Zhang *et al.*, "Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks," in *Proceedings of ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '15. New York, NY, USA: ACM, 2015, pp. 161–170.

[6] P. Meloni *et al.*, "NEURAghe: Exploiting CPU-FPGA Synergies for Efficient and Flexible CNN Inference Acceleration on Zynq SoCs," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 11, no. 3, pp. 1–24, 2018.

[7] S. I. Venieris and C.-S. Bouganis, "Latency-driven design for FPGA-based convolutional neural networks," *27th International Conference on Field Programmable Logic and Applications (FPL)*, pp. 1–8, 2017.

[8] H. Sharma *et al.*, "From high-level deep neural models to FPGAs," *49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 1–12, 2016.

[9] K. Guo *et al.*, "Angel-eye: A complete design flow for mapping CNN onto embedded FPGA," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, pp. 35–47, 2018.

[10] S. I. Venieris and C.-S. Bouganis, "fpgaConvNet: Mapping regular and irregular convolutional neural networks on FPGAs." *IEEE transactions on neural networks and learning systems*, 2018.

[11] J. T. Springenberg *et al.*, "Striving for simplicity: The All Convolutional Net," *CoRR*, vol. abs/1412.6806, 2015.