A Flexible Multilayer Perceptron Co-processor for FPGAs

Zeyad Aklah(✉) and David Andrews

Computer Science and Computer Engineering, University of Arkansas, Fayetteville, AR 72701, USA
{zaklah,dandrews}@uark.edu
http://www.hthreads.uark.edu

Abstract. Implementing a custom Artificial Neural Network (ANN) in hardware lacks the scalability and the flexibility of changing from one topology to another at run time. This paper presents a Multilayer Perceptron Co-processor (MLPCP) targeting FPGAs that is configurable during design time and programmable during run time. The MLPCP can be reprogrammed at run time to rapidly change network topologies and use different activation functions. This allows application developers to change parameters of a given network without the need to resynthesize. This also allows the MLPCP to be used for different applications during run time. Run time results show the MLPCP can deliver performance levels close to those of a custom ANN, and can execute network topologies that cannot fit into FPGAs with limited resources. Performance comparisons against software versions show up to 70x speedup compared to a MicroBlaze running at 100 MHz, and 4x compared to a Zynq running at 667 MHz.

1 Introduction

Researchers have been exploring the performance benefits of implementing custom Artificial Neural Networks (ANNs) within FPGAs [1][10][9][8][3][7]. Although FPGA densities are growing their finite set of resources will limit the size and accuracy of the ANN. Investigations have occurred to reduce the resource footprint required to implement a custom ANN [9],[1],[6].

However constructing a custom ANN requires that the topology of layers, as well as the types of arithmetic and activation functions used in the neural network be defined before synthesis. This prevents run time changes to the topology of the network as well as key parameters that effect accuracy such as arithmetic precision and types of activation functions implemented in the neural network[11]. Clearly it would be desirable to synthesize once, and then reuse the neural network under different configurations for different applications.

Esmaeilzadeh et al. [4] addressed this lack of flexibility by proposing a programmable neural network co-processor targeted for implementation as an Application Specific Integrated Circuit (ASIC). The co-processor is first optimized through design space exploration prior to fabrication. Once a design is selected

K. Sano et al. (Eds.): ARC 2015, LNCS 9040, pp. 427–434, 2015.
DOI: 10.1007/978-3-319-16214-0_39

and fabricated it can then sequence the execution of multiple topologies at run time.

This paper presents a configurable and programmable Multilayer Perceptron Co-Processor (MLPCP) for implementation on FPGAs that extends the configurability of the design reported in [4]. During the design phase the number of PEs, data representation (floating point or fixed point), Activation Function (AF) implementation approach (BRAM lookup tables or synthesized using Vivado HLS) can be configured by setting parameters. Once configured the MLPCP co-processor design automatically scales the register set and interconnect to support the number of PEs specified. Once synthesized the MLPCP is then programmable at run time to implement any topology (up to a set maximum) and use mixes of different types of activation functions. This provides the flexibility to change the precision and accuracy of results, or reuse the co-processor to support the needs of multiple application domains without having to resynthesize hardware.

The results in section(4) show the benefits as well as performance costs of this flexibility. Performance results show the flexibility and reprogrammability of the MLPCP do come at a modest decrease in peak performance compared to fully custom implementations of small to modest sized neural networks. Results then show the same MLPCP can continue to compute larger neural networks that exceed the resource limitations of the FPGA for a custom implementation.

2 MLPCP Architecture

Figure 1 shows the generic structure of an MLPCP core. The MLPCP architecture consists of a linear array of Processing Elements (PEs), a scheduler, controller, configuration registers, local memory and three external interface connections. Both input data and weights are transferred through the same input channel. The MLPCP

is automatically generated based on a set of configuration parameters provided by the user. The main parameters are the number of PE's, type of activation functions, and arithmetic precision. Designers can vary these parameters to explore area and performance tradeoffs for a particular FPGA. Once the parameters have been set, the core scales itself and can then be synthesized. Regardless of how the parameters are set, all configurations are still run time programmable and will support different topologies, number of neurons, and activation functions (up to a set maximum). Thus designers can perform cost performance tradeoffs to arrive at an MLPCP tailored for their FPGA and set of system requirements.

Processing Elements Figure 1.b shows a block diagram of the neuron PEs.

Each PE contains support for three types of AFs: a step function, log sigmoid, and tangent sigmoid. During the design phase, designers can choose to implement the AFs as either computed functions (synthesized hardware) or using

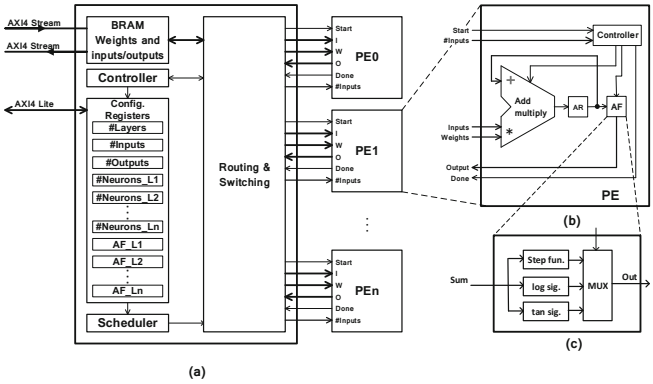

Fig. 1. MLPCP Architecture; (a). MLPCP structure; (b). a PE structure (AR: Accumulation Registers); (c). AF unit.

LUTs. Each layer can utilize different AFs programmed through the configuration registers at run time. This provides the flexibility to implement different networks at run time.

Scheduler The scheduler divides each layer into group(s) of neurons equal in size to the available number of PEs.

Thus for some topologies, the number of neurons in a layer is not divisible evenly by the number of PEs. In these cases, the remaining neuron(s) will be assigned to the first PE(s) during the next cycle. For example, with 8 PEs and 20 neurons in a layer, the scheduler will sequence two groups of eight neurons and one group of four. All neurons in a group are processed concurrently, with different groups processed sequentially. Based on the scheduler assignment, the controller aligns weights and inputs for each neuron in each PE's FIFO. Figure 2 illustrates how the scheduler computes neurons in an MLPCP with four PEs. The network is divided into 7 groups (G1 to G7). The scheduler then assigns each group sequentially onto the PEs. The outputs of each group within a layer are saved in the Layer Buffer, and assigned as inputs to the next layer.

Controller The controller is responsible for organizing weights and inputs for the neurons. During setup time, if AFs are not built in HW (synthesized), the controller reads the AFs values throughout the streaming input channel and stores them in PEs' local memory. Also, it calculates the number of weights that will be streamed to the MLPCP based on the configured registers. The controller divides the weight matrix into groups equal to the number of PEs and aligns them into each PE's FIFO. The controller also aligns the outputs of each layer with the appropriate weights to be used in the next layer. At the output layer, the controller streams the results out of the programmable MLPCP and generates "last" signal.

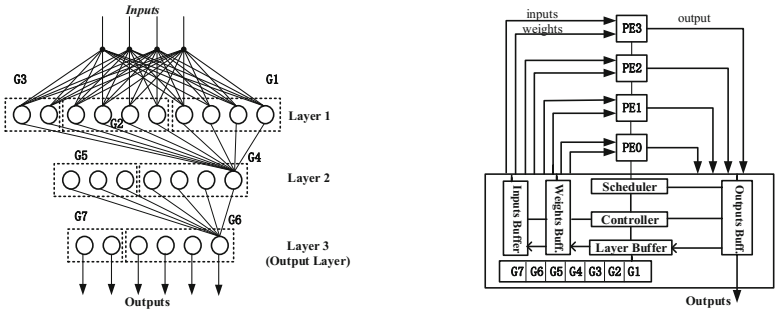

Fig. 2. Scheduling neurons in MLPCP with four PEs

Configuration Registers The following programmable registers are provided:

- #Layers: number of layers in the network.
- #Inputs: number of inputs.
- #Outputs: number of outputs.
- #Neurons_L1: number of neurons in first layer.
- #Neurons_L2: number of neurons in second layer.
- #Neurons_Ln: number of neurons in n^{th} layer.
- #AF_L1: the type of activation function in first layer.
- #AF_L2: the type of activation function in second layer.
- #AF_Ln: the type of activation function in n^{th} layer.

The number of registers scales based on the maximum network settings such as: Maximum Number of Layers (MNL), Maximum Number of Neurons (MNN), Maximum Number of Neurons in Largest Layer (MNNLL), Maximum Number of Inputs (MNI), and Maximum Number of Outputs (MNO).

3 Evaluation and Results

The MLPCP was written in C++ and generated using Xilinx's Vivado-hls 14.2 tools. We configured, synthesized and ran four versions of the MLPCP and compared their performance against software implementations for three applications. Comparisons were performed against a 660 MHz diffused ARM processor on the Zedboard(xc7z020clg484-1) and a 100 MHz soft IP MicroBlaze processor.

Performance results for these tests are presented in Section 3. We then implemented two custom neural network versions of three different test applications on the Zedboard for size comparisons. These results are presented in Section 3.1.

MLPCP Configurations: The following four versions of the MLPCP were used in our evaluations:

1. MLPCP-1: Four PEs, arithmetic floating point, AFs: synthesized (Vivado-Hls).

Table 1. Execution times for software (MicroBlaze, ARM) and MLPCP-x

Func.	Network		SW(μs)		MLPCP-1 run time(μs)	MLPCP-2 run time(μs)	MLPCP-3 run time(μs)	MLPCP-4 run time(μs)
	Order	Topology	Microbl.	ARM				
Sched.	1	9,2,6	52.36	2.52	5.91	5.24	5.04	4.6
	2	9,4,6	78.43	4.82	6.5	5.7	5.42	4.9
	3	9,16,8,6	296.49	17.84	17.63	11.22	12.22	8.7
	4	9,40,6	449.83	30.40	29.8	17.96	19.54	13.72
	5	9,12,27,6	510.91	31.54	29.8	17.96	20.22	15.93
JPEG	6	64,16,64	1319.48	94.86	83.66	50.31	58.45	39.36
	7	64,32,16,64	2848.89	158.9	137.93	81.58	93.19	60.74
	8	64,32,16,32,64	4021.37	279.62	190.2	112.67	125.91	80.15
	9	64,32,16,48,40,16,64	6682.37	411.32	213.41	133.84	143.32	95.52

2. MLPCP-2: Eight PEs, arithmetic floating point, AFs: synthesized (Vivado-Hls).
3. MLPCP-3: Four PEs, arithmetic fixed point, AFs: lookup table in BRAM (100 samples with 8-bit resolution).
4. MLPCP-4: Eight PEs, arithmetic fixed point, AFs: lookup table in BRAM (100 samples with 8-bit resolution).

Both MLPCP-3 and MLPCP-4 used (sign=1,word size=8,fraction=4) inputs, (1,17,8) weights, (1,29,16) weighted sum, and (1,8,4) outputs in fixed point data representation. The configuration parameters were set to the following values to bound the maximum network size that each MLPCP would run: MNL=6, MNN=348, MNNLL=64, MNI=64, MNO=64.

Test Applications Three applications were used in our evaluations. The first is inversek2j an inverse kinematics application for a 2-joint arm[4].

The second is an adaptive smart-scheduler reported in [2]. The third is a standard JPEG encoder. Each application was first trained offline on a desktop PC in MATLAB. Different topologies were evaluated during the training phase. The topologies used in this study are not guaranteed to provide the best efficiency and accuracy.

Performance Evaluation The four versions of the MLPCP were evaluated against software implementations on both a MicroBlaze and ARM .The software source code could be found in [5]. It was modified and optimized to run on embedded processors. Data and instruction caches were turned on for both processors. Input data sets were transfered using DMA into local BRAMs for all test cases (processors and MLPCPs). The reported time in Table 1 included transfer and execution time.

Table 1 lists execution times for the smart scheduler and JPEG encoder applications trained using nine different topologies. Each of the four MLPCP systems were programmed at run time to implement each of the topologies. For a given topology, the first number represents the number of inputs. The

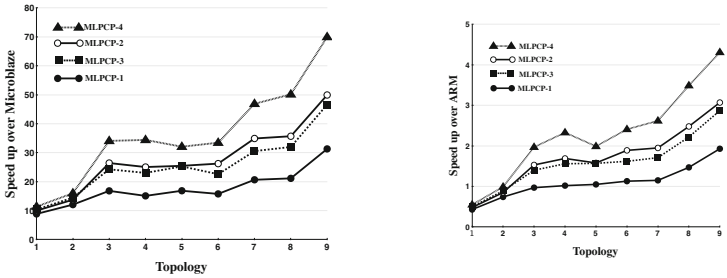

Fig. 3. Speed up over software on MicroBlaze (left) and ARM (right) processors

cardinality of the $n-1$ remaining integers represent the number of layers, with each integer representing the number of neurons in that particular layer.

The execution times from Table 1 are replotted in Figure 3 to show the speed up over software implementations run on the MicroBlaze and ARM processors respectively.

The MicroBlaze was running at 100 MHz, the same clock frequency as the co-processor. The ARM however, was running at 667 MHz, 6.6x faster than the 100 MHz MLPCP. Even with the 6.6x advantage, the MLPCP begins outperforming the ARM at topology (3); a relatively small and simple neural network with three layers and 30 neurons. In both cases, speedups generally increase as the size and complexity of the topology increases.

3.1 Resource Comparisons

Generalization typically comes with performance and resource efficiency costs compared to customization. This analysis shows the performance and resource efficiency costs associated with the programmable MLPCP. We quantified these costs by building and then comparing the following two custom designs with our four MLPCP systems.

Custom hardware Designs: The following two custom hardware design approaches are from [1],[10] and [6]:

1. CNN1: A fully parallel network (the number of PEs equal to the number of neurons in the network) with pipelining between layers.
2. CNN2: A network with the number of PEs equal to the number of neurons in the largest layer. Each layer was then multiplexed on the array of PE's.

Table 2 shows the resource usage for our four MLPCP systems on the Zedboard.

Comparisons Between MLPCP-4 and CNNx : For comparisons we implemented the two custom networks on the Zedboard. One specific topology was used for each application and then implemented using the CNN1 and CNN2

Table 2. Resource Utilizations on Zedboard

Version	PEs	AF	BRAM_18k	DSP48E	FF	LUTs
MLPCP-1	4	Synth	8.9%	37%	6.59%	16.21%
MLPCP-2	8	Synth	13.2%	74.09%	12.13%	31.15%
MLPCP-3	4	LUTs	4%	4%	0.2%	3.2%
MLPCP-4	8	LUTs	8.2%	7.7%	2.9%	4.9%

Table 3. Resources Comparisons for MLPCP-4 and CNNs on Zedboard

Function	Design	PEs	BRAM_18k	DSP48E	FF	LUTs	Performance (μs)
Inversek2j (2,4,2)	CNN1	6	0	2.2%	1.1%	6%	2.29
	CNN2	4	0	1.3%	0.9%	4.9%	2.49
	MLPCP-4	8	7.5%	7.2%	2.9%	4.9%	3.86
Scheduler (9,10,6)	CNN1	16	0.3%	13.6%	4.2%	16.32%	3.8
	CNN2	10	0.3%	9.0%	3.4%	12.6%	4.4
	MLPCP-4	8	7.5%	7.2%	2.9%	4.9%	5.9
JPEG (64,8,64)	CNN1	72	25.7%	37.7%	15.4%	122%	-
	CNN2	64	22%	30%	11%	100%	-
	MLPCP-4	8	7.5%	7.2%	2.9%	4.9%	25.67

approaches. Inversek2j was first trained and then implemented using fixed point arithmetic and lookup tables for the AFs. The MLPCP-4 system was chosen for comparison.

Several interesting results can be drawn from the comparisons in Table 3. The inversek2j application represents a very small custom neural network with fewer than the eight PEs contained within MLPCP-4. Thus the MLPCP-4 includes two PEs and additional control and sequencing logic not used. For this case the CNNs were more resource efficient. The smart scheduler and JPEG applications show the MLPCP-4 becoming more resource efficient as the size of the neural network exceeds its 8 PEs. Both custom designs showed how the resource requirements grow with size of the network topology compared to the MLPCP-4. It can be seen in Table 3 that the custom implementations for the JPEG application exceeded the LUT resources available on the Zynq and could not be synthesized.

Table 3 shows the performance cost of programmability. As would be expected a fully custom, pipelined implementation (CNN1) outperformed the custom design that limited PEs to the largest layer (CNN2) as well as a very general and programmable co-processor (MLPCP-4).

4 Conclusion

This paper presented a new Multilayer Perceptron Co-Processor (MLPCP) designed for implementation on FPGAs. The MLPCP provides the advantages of reuse and scalability over custom designed ANNs. Once synthesized for a

particular FPGA, the MLPCP can be configured through a programmable set of registers at run time to assume different topologies and achieve different accuracy of results. This allows the MLPCP to be used for different sets of requirements and across different applications.

References

1. Canas, A., Ortigosa, E., Ros, E., Ortigosa, P.: FPGA implementation of a fully and partially connected MLP. In: Omondi, A., Rajapakse, J. (eds.) FPGA Implementations of Neural Networks, pp. 271–296. Springer, US (2006). http://dx.doi.org/10.1007/0-387-28487-7_10
2. Cartwright, E., Sadeghian, A., Ma, S., Andrews, D.: Achieving portability and efficiency over chip heterogeneous multiprocessor systems. In: Proc. of the 24th Intl. Conf. on Field Programmable Logic and Applications (FPL), pp. 1–4 (2014)
3. Cheung, K., Schultz, S., Luk, W.: A large-scale spiking neural network accelerator for FPGA systems. In: Villa, A., Duch, W., Érdi, P., Masulli, F., Palm, G. (eds.) ICANN 2012, Part I. LNCS, vol. 7552, pp. 113–120. Springer, Heidelberg (2012). http://dx.doi.org/10.1007/978-3-642-33269-2_15
4. Esmaeilzadeh, H., Sampson, A., Ceze, L., Burger, D.: Neural acceleration for general-purpose approximate programs. In: 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), pp. 449–460, December 2012
5. Gure, A.: Multilayer perceptron neural network in c. https://github.com/sanjeevk001/workingfiles/tree/master/mlp-bp-fxp-5
6. Himavathi, S., Anitha, D., Muthuramalingam, A.: Feedforward neural network implementation in FPGA using layer multiplexing for effective resource utilization. IEEE Trans. on Neural Networks **18**(3), 880–888 (2007)
7. Jung, S., Kim, S.S.: Hardware implementation of a real-time neural network controller with a DSP and an FPGA for nonlinear systems. IEEE Transactions on Industrial Electronics **54**(1), 265–271 (2007)
8. Krips, M., Lammert, T., Kummert, A.: FPGA implementation of a neural network for a real-time hand tracking system. In: Proc. of the 1st Intl. Workshop on Electronic Design, Test and Applications, pp. 313–317 (2002)
9. Misra, J., Saha, I.: Artificial neural networks in hardware: A survey of two decades of progress. Neurocomputing **74**(13), 239–255 (2010). Artificial Brains http://www.sciencedirect.com/science/article/pii/S092523121000216X
10. Moussa, M., Areibi, S., Nichols, K.: On the arithmetic precision for implementing back-propagation networks on FPGA: a case study. In: Omondi, A., Rajapakse, J. (eds.) FPGA Implementations of Neural Networks, pp. 37–61. Springer, US (2006). http://dx.doi.org/10.1007/0-387-28487-7_2
11. Ortega-Zamorano, F., Jerez, J., Franco, L.: FPGA implementation of the c-mantec neural network constructive algorithm. IEEE Trans. on Industrial Informatics **10**(2), 1154–1161 (2014)