Design and Performance Comparison of CNN Accelerators based on the Intel Movidius Myriad2 SoC and FPGA embedded prototype

Angelos Kyriakos[1], Elissaios-Alexios Papatheofanous[1], Bezaitis Charalampos[1],
Evangelos Petrongonas[2], Dimitrios Soudris[2] and Dionysios Reisis[1]
[1]Electronics Lab, Physics Department, GR-15784,
National and Kapodistrian University of Athens, Greece
[2]Mircrolab, School of Electrical & Computer Engineering,
National Technical University of Athens, Greece
Email: {akyriakos, eapapatheo, bezaitisc, dreisis}@phys.uoa.gr, {vpetrog, dsoudris}@microlab.ntua.gr

***Abstract*—Evolving Convolutional Neural Networks (CNNs) and their execution time performance are key factors for a wide range of applications that are based on deep learning. The need for meeting the applications' time constraints led to design AI accelerators and the current work contributes to this effort by presenting CNN accelerators based on two different design approaches: a) developing CNNs on a power efficient System on Chip (SoC), the Myriad2 and b) a VHDL application specific design and the corresponding FPGA architecture. Both systems target the optimization of time performance regarding the MNIST dataset application. The paper describes the two systems and compares the performance results.**

***Index Terms*—Neural Networks, Accelerator, Myriad2, FPGA**

I. Introduction

During the past decades the progress of the CNN algorithmic techniques has led to a significant improvement of the classification effectiveness for certain tasks including the image recognition. A plethora of applications that involve these CNN techniques impose restrictions on the execution time of the task and hardware accelerators were introduced to improve the time efficiency of the tasks. Consequently, researchers and engineers are involved in an ongoing effort to upgrade the performance by proposing novel accelerator designs that are based on SoCs, GPUs, FPGAs and multi-core systems [1], [2] , [3], [4].

Aiming at contributing to this effort the current work designs and presents two solutions for CNN accelerators, analyzes their advantages and compares their performance. The comparison of the two accelerators is based on their optimization for the MNIST dataset. The first accelerator exploits the features of the Intel Movidius Myriad2 SoC and the second is a VHDL design implemented on a Xilinx Kintex Ultrascale (xcku060 - ffva1156, -1 low power speed grade). The choice of these two systems was based on their characteristics that allow them to distinguish among others. The first was the power efficiency of the Myriad2 allowing the architecture designers based on this SoC to develop capable CNN computing systems with low energy consumption requirements [5]. The second characteristic is the advantage of the FPGA as a development environment that allows the designers to configure the parallelism degree and optimize the VHDL design, elements that both lead to advance the execution time of the target application [6]. A notable example of such a comparative study is the work in [2] comparing FPGA implementations to Intel Neural Compute Stick.

The present work was motivated by the need of employing CNN accelerators in environments where energy consumption is imperative to the entire system's specifications, while the required number of these systems cannot justify the ASIC developmnent. This reason initiated the study for the architectural design trade-off, that shows an almost 16x performance advantage of the FPGA VHDL implementation of the CNN accelerator over the bare metal development (without Real-Time Executive for MultiprocessorSystems (RTEMS)) of the CNN on the Myriad2 SoC. Considering the power consumption, the Myriad2 requires 5 times less power than the FPGA accelerator to perform the same CNN calculations. Along with these results it is worth noting that, in order to achieve these performance results in the FPGA, increased development effort is required for the detailed VHDL design. On the other hand, the software development of the CNN on Myriad2 SoC although it requires experienced designers, it is a less time consuming procedure and provides the benefit of simpler reconfiguration.

The paper is organized with the following Section presenting the CNN software model. Sections III and IV describe the accelerator designs based on the Myriad2 SoC and the FPGA. Section V analyzes and compares the two accelerators performance and finally, Section VI concludes the paper.

II. CNN Software Model

The application considers the MNIST dataset [7] dedicated to recognize what digit $(0, \ldots, 9)$ is printed in a handwritten figure.

978-1-7281-3572-4/19/$31.00 ©2019 IEEE
DOI 10.1109/ICCAIRO47923.2019.00030

A. Model Architecture and Training

The model was trained with the MNIST dataset containing 60K total handwritten digit grayscale images with dimensions 28x28 pixels. Out of the 60K images, 50K where selected for the training process and the remaining 10K were used for the model validation.

The dataset images were given as input to the CNN depicted on Fig. 1, which was implemented via the TensorFlow Estimator API consisting of the following operations:

1) one convolution layer containing 32 filters of 5×5 kernel size each,
2) a max pooling layer dedicated to downsample the feature maps by extracting the max value of 2×2 windows with stride 2,
3) a 30-neuron fully connected layer and finally, d) a 10-neuron output layer; this output result represents the final classification.

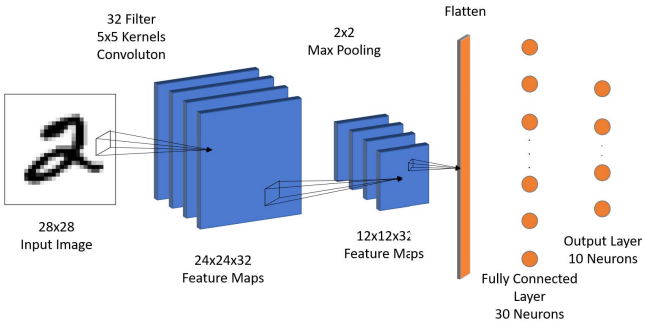

Fig. 1. Model Architecture

The implementation results showed that the accuracy of the model was not affected by either using or not using padding. Consequently, we opted to avoid any padding for our model in the convolution layer and thus, the CNN design minimized the unnecessary computations both in Myriad2 and in the FPGA.

The model uses the ReLU activation function [8] in all the layers of the model's architecture apart the output layer. The latter choice was taken because ReLU provides the best performance with respect to the computational cost ratio and it is efficiently implemented in hardware regarding the resources' utilization. The output layer calculates the argmax and softmax of the output layer's tensors in order to produce the classification. The model uses the Adam Optimization Algorithm [9] with input from the cross-entropy loss function and it achieves 98.66% accuracy after 9 epochs.

B. Inference Software Implementation

We developed a C software program that calculates the Inference of the CNN model. The C software uses the same weights and parameters of the TensorFlow application and calculates the inference in the following steps:

(a) It calculates the convolution of the input image with the 1st filter's kernel.
(b) It adds the corresponding bias and applies the ReLU function.
(c) To the resulting array it performs the max pooling operation.
(d) It repeats the steps a, b, c until it calculates the feature map of each filter of the Convolution Layer, by calling the corresponding function in each iteration.
(e) It calculates the matrix multiplication of the Fully Connected Layer: one neuron at a time. It calculates the dot product of the feature map array with the weight array of each neuron of the Fully Connected Layer.
(f) Finally, it calculates the matrix multiplication of the output layer resulting in the classification of the network. The results are not normalized into a probability distribution, so that we can directly compare these with the corresponding FPGA results.

III. Myriad2 CNN Accelerator

Myriad2 is a multicore System on Chip (SoC) that constitutes a Visual Processing Unit (VPU) [10]. The Myriad2 SoC's architecture is depicted on Fig. 2. The VPU incorporates 12 SHAVE processors optimized for machine learning applications with very long instruction word (VLIW) that enable the parallellization of neural network algorithms (e.g. inference). Moreover, the Myriad2 SoC includes a multicore on-chip memory subsystem, a shared 2MB memory block called Connection Matrix (CMX) memory, and 512MB low-power DDR3 DRAM. Furthermore, the Myriad2 SoC incorporates 2 Leon processors in order to support real time operating systems and various peripheral devices. The major advantage of the device is that it performs at 600 MHz, up to a trillion operations per second, with power consumption lesser than 1 W [10].

A. Parallel Software Implementation

The design of the Myriad2 CNN Accelerator is revolved around the parallel architecture of the SoC. The design utilizes all the 12 available SHAVE processors in order to reduce the inference execution time and increase the throughput of the accelerator. The Myriad2 Accelerator is designed under the bare metal programming paradigm, which allows the use of the Leon processors without any operating system and with minimal schedulers to control the pipeline of applications. The accelerator incorporates one Leon processor (OS) operating as the scheduler of the 12 SHAVEs. The Myriad2 Accelerator omits the use of RTEMS to eliminate the operating system overhead in the cost of integration efforts for the developer.

The Myriad2 CNN Accelerator is efficient regarding the utilization of the available memory resources of the SoC. Given the fact that the application has low memory requirements, our effort targeted the use of the CMX memory for storing all the neural network parameters (weights, biases) and the input image and thus, we avoided the performance degradation caused by the use of the DRAM. Moreover, the SHAVE code can be stored either in the CMX memory or in the DRAM. The Myriad2 Accelerator takes advantage of this feature, by storing

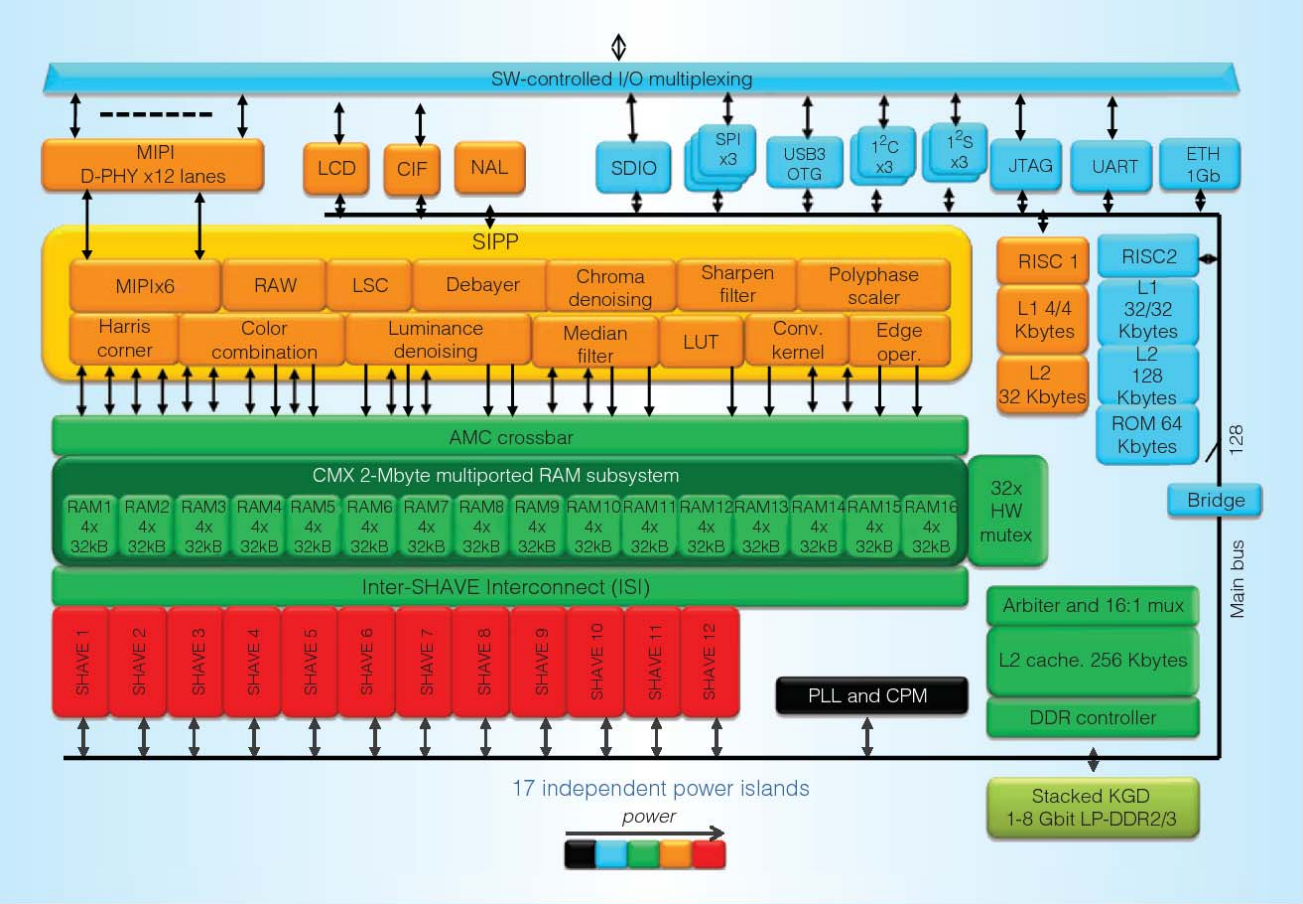

Fig. 2. Myriad2 Architecture

the SHAVE code in the DRAM memory and the application data are stored in the CMX. This division of the memory provides a major performance gain because the program data reside in the fastest possible memory in the SoC. The fact that the program code resides in the DRAM memory proves not to be a bottleneck, because we are able to utilize the caching capabilities of the Myriad2 SoC reducing the DRAM performance penalty when fetching instructions from it.

The mapping of the C software implementation to the 12 SHAVE processors, operating in parallel, is as follows:

(a) The computations of the 32 filters of the Convolution Layer and the corresponding max pooling operations are mapped onto the available processors on a per filter basis (e.g. when the 32 filters are mapped onto 12 SHAVEs: each of the first 8 SHAVEs performs the convolution of 3 filters and each of the last 4 SHAVEs performs the convolution of 2 filters).

(b) The completion of the calculations of the Convolution and Pooling Layers constitutes an execution barrier, reaching which, indicates that the input of the Fully Connected Layer is available. Each of the 30 neurons of the Fully Connected Layer is mapped onto a SHAVE processor: each of the first 6 SHAVEs performs 3 dot products and each of the last 6 SHAVEs performs 2.

(c) Finally, when all SHAVEs finish the execution of the Fully Connected layer, this constitutes a second barrier, then the calculation of the values of the 10 neurons of the Output Layer will be completed.

The organization of the SHAVE data in the CMX memory is critical to the performance of the accelerator as well as the power consumption of the accelerator. Targeting the reduction of the data sharing and consequently the memory access clashes among the SHAVEs, we opted for data redundancy in the convolution layer by storing multiple copies of the convolution parameters as many as the number of SHAVE processors: one copy of the input image, the convolution layer's weights and biases is stored in each SHAVE's CMX space. The weights of the Fully Connected Layer are stored in a common access region of the CMX memory, due to their size that restricts the data redundancy.

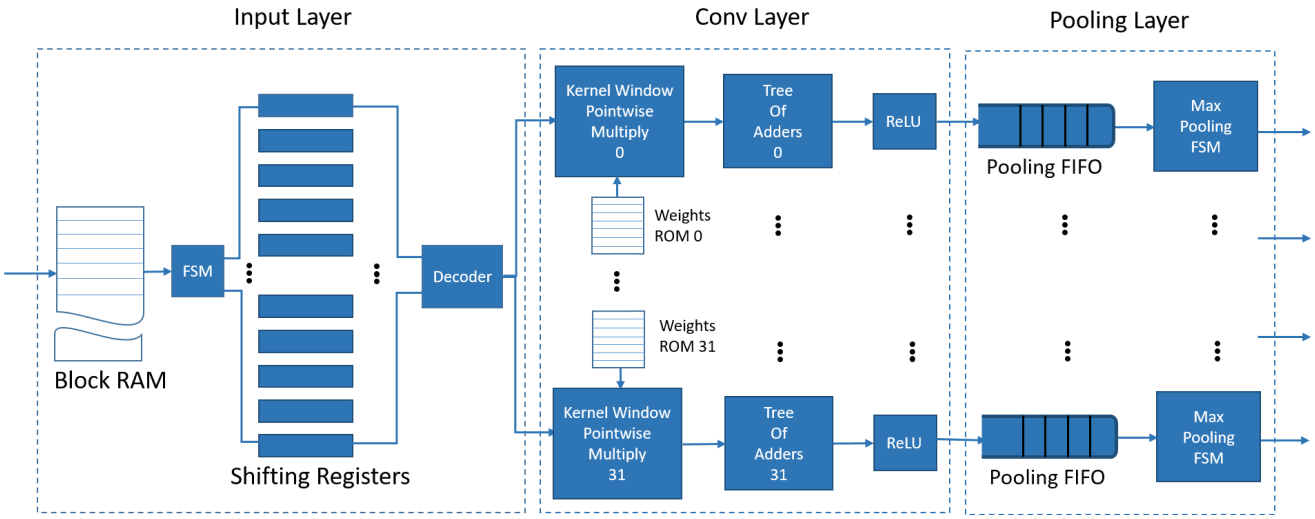

Fig. 3. FPGA Architecture: Input, Convolution & Pooling Layers

IV. FPGA Accelerator

The high performance FPGA CNN Accelerator follows the design approach of the architecture to that presented in [11] targeting the same CNN application with the difference that in the current work, the design includes modifications that keep the performance but they also target the accuracy and the power consumption improvement. For these reasons the Xilinx Kintex Ultrascale xcku060 has been chosen. The architecture of the FPGA CNN Accelerator corresponds to the Software model and it is depicted in Fig. 3 and 4. We also developed an arithmetic bit accurate (BAM) model in Matlab, that operates with fixed-point integer representation of numbers (Q5.14), to validate the FPGA design. The architecture of the FPGA CNN Accelerator includes the implementation of 5 layers: the Input Layer, the Convolution Layer, the Pooling Layer, the Fully Connected Layer and the Output Layer. The Input Layer stores the input image and performs the sliding window algorithm of the 2-D convolution on the input image, by utilizing 2 sets of shifting registers. The Convolution and Pooling Layers consist of 32 parallel operating blocks, each of which is performing the calculation of a single convolution filter in parallel and it is highly pipelined. The Fully Connected Layer's matrix multiplication includes the execution of 30 Vector Multiplications and for each multiplication there a dedicated corresponding Vector Multiplier (shown in Fig. 4) so that all the multiplications are executed in parallel. The Output Layer performs the final calculations that produce the classification.

The architecture of the FPGA Accelerator exploits the parallelization that the CNN model provides and is highly pipelined in order to increase throughput. Note also that, the use of pipelining in the design leads to fewer glitches because it has fewer logic levels between registers and therefore, improves the power consumption [12]. For a single input image the FPGA CNN accelerator computes the classification result in 4938 clock cycles. Moreover, the VHDL implementation of the FPGA CNN Accelerator utilizes only on-chip memory, a fact that leads to increased performance and reduced power consumption [13]. Furthermore, the computations are all implemented with fixed-point integer arithmetic that induces negligible error with respect to the TensorFlow model. In order to achieve the objective performance the FPGA CNN accelerator has been designed as two distinct parts with respect to the operating frequency: the Convolution and Pooling Layers perform at 200 MHz while the Fully Connected and Output Layers at 240 MHz. The architecture utilizes 10.34% of the FPGA logic, 30.16% DSP Slices and 15.28% of block-RAM memory resources of the Kintex Ultrascale xcku060.

V. Performance & Power Comparison

The current section compares the two systems. The comparison considers the execution of the classification on a single image. Table I presents the Myriad2 parallel inference implementation results, where the CNN design on the Myriad2 shows an almost linear speedup with respect to the number of SHAVE processors employed in the computations. Moreover, Fig. 5 presents in a single chart the relation of the speedup, power consumption and execution time of the Myriad2 parallel software implementation with respect to the number of SHAVE processors that are utilized in the design. We can observe that, there is a small increase in the power consumption in relation with the number of SHAVE processors, which is almost 15mW for each additional SHAVE processor. Therefore, the Myriad2 CNN accelerator is advantageous because increasing the number of used SHAVEs results in higher speedup at a small penalty in the power consumption. Furthermore, Table II shows an almost 0.1 ms performance edge of the bare metal

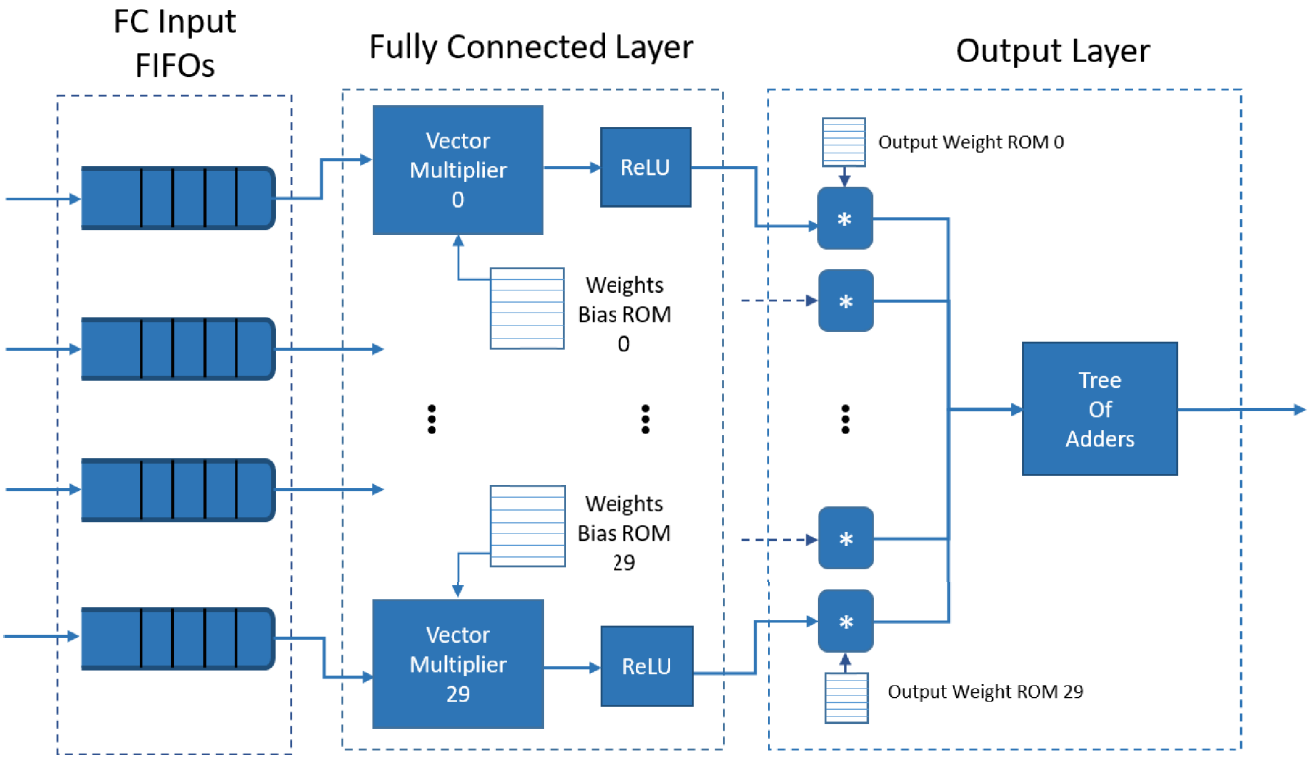

Fig. 4. FPGA Architecture: Fully Connected & Output Layers

implementation compared to the RTEMS. The advantage of the bare metal implementation is more evident in the designs utilizing more SHAVE processors, where the execution time of the CNN accelerator is in the same order of magnitude with the RTEMS overhead.

Table III shows a 16x performance advantage of the FPGA VHDL CNN accelerator compared to the bare metal implementation (without RTEMS) of the CNN on the Myriad2 SoC. On the other hand, comparing the power consumption, the Myriad2 requires 5 times less power than the FPGA accelerator for completing the same CNN calculations. Finally, comparing the development time and effort that is taken on the two development platforms we have to note that, first the CNN accelerator design process on the Myriad2 is time efficient compared to the design and development on the FPGA. On the other hand, the VHDL design allows for improvements leading to increased performance.

TABLE I
MYRIAD2 PARALLEL IMPLEMENTATION SPEEDUP

# Shaves	*1*	*2*	*4*	*8*	*12*
Exec. Time (ms)	1.82	1.04	0.54	0.39	0.35
Speedup	-	1.75	3.37	4.66	5.2
Power Consumption (mW)	510	529	563	636	707

TABLE II
MYRIAD2 PROGRAMMING PARADIGM PERFORMANCE COMPARISON

Execution Time (ms)	
RTEMS	***Bare Metal***
0.44	0.35

[a]Implementation with 12 SHAVEs.

TABLE III
MYRIAD2 - FPGA ACCELERATOR PERFORMANCE - POWER COMPARISON

	Execution Time (ms)	*Power Consumption (mW)*
Myriad 2	0.35	707
FPGA	0.021	3631

VI. CONCLUDING REMARKS

The current paper has presented two CNN accelerator designs based on the Intel Myriad2 SoC, which is optimized for low power image processing applications, and a VHDL architecture implemented on an FPGA. The comparison of the two designs shows that the VHDL FPGA accelerator has a significant advantage with respect to the CNN execution time while the Myriad2 SoC prevails when the power consumption becomes the key element of the design. Also note that, to optimize the performance in the FPGA, the designers have to give a significant amount of effort, while developing the CNN

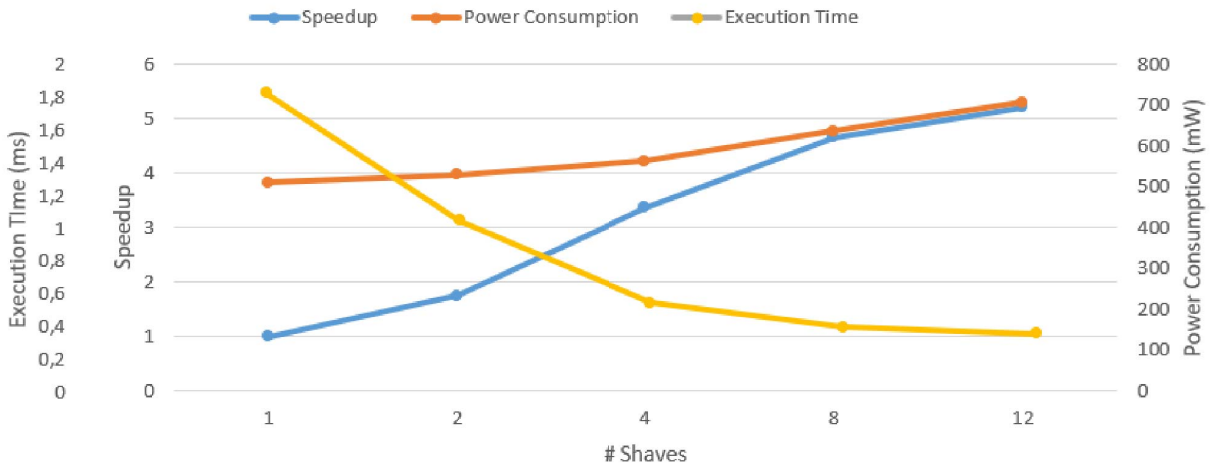

Fig. 5. Myriad2 Parallel Implementation Speedup, Power and Execution Time

on the Myriad2 SoC, is a quite faster process.

Future work includes the design and implementation of more complicated CNN based applications for the Myriad2 SoC that gained lately the designers' attention due to the power consumption efficiency. Our first goal is the enhancement of the software design, that is to support applications requiring larger CNN model with more convolution layers. The second objective is to study and propose techniques for the effective utilization of DRAM memory on Myriad2. Another significant result will be the use of DMA driver for the communication between the CMX and DRAM. Finally, considerably improvement with respect to execution time maybe achieved by including assembly code in the current implemented C programs. Considering our future work on the FPGA platforms, we will study the feasibility of including larger CNN models with more convolution layers. Moreover, an important issue is to optimize the utilization of the on-chip memory for the CNN operation and use the external DRAM memory only for those data that don't affect the CNN execution time.

ACKNOWLEDGEMENT

The research work was supported by the Hellenic Foundation for Research and Innovation (HFRI) under the HFRI PhD Fellowship grant (Fellowship Number: 29).

REFERENCES

[1] T. Wang, C. Wang, X. Zhou, and H. Chen, "A survey of FPGA based deep learning accelerators: Challenges and opportunities," *CoRR*, vol. abs/1901.04988, 2019. [Online]. Available: http://arxiv.org/abs/1901.04988

[2] Gianmarco Dinelli, Gabriele Meoni, Emilio Rapuano, Gionata Benelli, and Luca Fanucci, "An fpga-based hardware accelerator for cnns using on-chip memories only: Design and benchmarking with intel movidius neural compute stick," *International Journal of Reconfigurable Computing*, vol. 2019, Article ID 7218758. [Online]. Available: https://doi.org/10.1155/2019/7218758

[3] R. A. Solovyev, A. A. Kalinin, A. G. Kustov, D. V. Telpukhov, and V. S. Ruhlov, "Fpga implementation of convolutional neural networks with fixed-point calculations," *CoRR*, vol. abs/1808.09945, 2018.

[4] J. H. Kim, B. Grady, R. Lian, J. Brothers, and J. H. Anderson, "Fpga-based cnn inference accelerator synthesized from multi-threaded c software," in *2017 30th IEEE International System-on-Chip Conference (SOCC)*, Sep. 2017, pp. 268–273.

[5] F. Tsimpourlas, L. Papadopoulos, A. Bartsokas, and D. Soudris, "A design space exploration framework for convolutional neural networks implemented on edge devices," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2212–2221, Nov 2018.

[6] K. Abdelouahab, M. Pelcat, J. Sérot, and F. Berry, "Accelerating CNN inference on fpgas: A survey," *CoRR*, vol. abs/1806.01683, 2018. [Online]. Available: http://arxiv.org/abs/1806.01683

[7] Y. LeCun and C. Cortes, "MNIST handwritten digit database," *http://yann.lecun.com/exdb/mnist/*, 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[8] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical evaluation of rectified activations in convolutional network," *arXiv preprint arXiv:1505.00853*, 2015.

[9] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[10] B. Barry, C. Brick, F. Connor, D. Donohoe, D. Moloney, R. Richmond, M. O'Riordan, and V. Toma, "Always-on vision processing unit for mobile applications," *IEEE Micro*, vol. 35, no. 2, pp. 56–66, Mar 2015.

[11] A. Kyriakos, V. Kitsakis, A. Louropoulos, E. Papatheofanous, I. Patronas, and D. Reisis, "High performance accelerator for cnn applications," in *2019 29th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, July 2019, pp. 135–140.

[12] S. J. E. Wilton, S.-S. Ang, and W. Luk, "The impact of pipelining on energy per operation in field-programmable gate arrays," in *Field Programmable Logic and Application*, J. Becker, M. Platzner, and S. Vernalde, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 719–728.

[13] J. Lamoureux and W. Luk, "An overview of low-power techniques for field-programmable gate arrays," in *2008 NASA/ESA Conference on Adaptive Hardware and Systems*, June 2008, pp. 338–345.